Conceptual Modeling for Computerized Information Systems Support in Organizations

John Krogstie
Information Systems Group
Faculty of Electrical Engineering and Computer Science
The Norwegian Institute of Technology
The University of Trondheim

December 21, 1995

Abstract

Organizations are continuously under the pressure of change, and develop through a process of *social construction*, based on its individuals' current perception of the world. Most organizations have a partly computerized information system to support adequate action so that perceived value addition may be achieved within this environment.

It is regarded that the evolutionary aspects of computerized information systems support are insufficiently covered by traditional approaches and tools. In addition to this, the process of social construction of the organizational reality is neglected in most development methodologies.

In this thesis, we have developed a methodological framework coined *system devtenance* for the application of *conceptual models* in developing and maintaining portfolios of computerized information systems in organizations, taking continuous evolution and social construction into account.

A survey-investigation has been performed among Norwegian organizations to get a better impression of current methodologies and perceived problems with computerized information systems support in organizations. The results from the investigation are compared with previous investigations of this kind to assess the more general applicability of our results. In addition the overall methodology is based on existing work within research and practice. The conceptual modeling framework is based on the frameworks in PPP and Tempora, which are extended with improved support for the modeling of rules, actors, roles, and communication. These extensions have been refined through their use in several case studies.

As an extension of earlier work, a framework for discussing the *quality* of conceptual models is created based on social construction theory. The framework is used for evaluation of approaches to support computerized information systems applying conceptual modeling techniques. The framework is also used in the evaluation of our own contributions.

On a high level of abstraction, the methodological framework covers all the important aspects for supporting computerized information systems in organization. Only the aspects related to conceptual modeling are described in detail in the thesis. The conceptual framework can be used for modeling according to all the main perspectives identified in literature, but not surprisingly, it has still room for improvements. Most aspects of model quality can also potentially be supported, although this is dependant on the implementation of additional tool-support and further practical experience to create a way of working that will support the creation of conceptual models of high quality.

Although much work is left to be done to support all the activities of system devtenance, we have in this thesis laid the foundation of what we feel is an important direction in the support of information systems in organizations using conceptual modeling techniques.

Preface

> *"The reason that I might see further is because I am standing on the shoulders of giants"*
>
> *Isaac Newton*

About the Thesis

This is a doctoral thesis submitted to the Norwegian Institute of Technology (NTH) for the doctoral degree ("doktor ingeniør"). The work has been carried out at the Information Systems Group, Department of Electrical Engineering and Computer Science, at the Norwegian Institute of Technology (NTH), under the supervision of Prof. Dr. Arne Sølvberg.

The Ph.D.study was funded by The Norwegian Research Council (NFR).

Problem Statement

Organizations are continuously under the pressure of *change* from both internal and external forces. Most organizations of some size are supported by and depend upon a *portfolio of application systems* that likewise have to be changed, often rapidly, for the organization to be able to keep up and extend its activities. The portfolio usually consists of a set of individual, highly integrated application systems whose long-term evolution should be looked upon in a coordinated manner.

Organizations are made up of individuals who perceive the world differently from each other. We claim that organizations develop through a process of *social construction*, based on its individuals' perception of the world. Parts of their local reality is externalized as organizational reality, i.e. into something that the social actors of the organization have to relate to in their work.

When developing and maintaining computerized information systems, *conceptual models* are constructed as part of the problem analysis and requirements specification process and are often used as an outset for the implementation of the application system. When developing models of an existing or future situation as it is perceived, we claim that social construction mechanisms should be taken into account, and support for discussing and consolidating incompatible models should exist.

The evolutionary aspects of computerized information systems support are insufficiently covered by traditional approaches and tools. In addition to this, the process of social construction of the organizational reality is neglected in most development methodologies.

Objectives

The main objectives of this research is to develop a methodological framework for the use of conceptual models in developing and maintaining portfolios of computerized information systems in organizations, taking the problems of continuous evolution and social construction into account.

To counter the traditional distinction between development and maintenance, the framework is coined with the phrase *systems devtenance.*

Approach

A survey-investigation has been performed among Norwegian organizations to get a better impression of current methodologies and problems within computerized information systems support in organizations. The results from the investigation are compared with previous investigations of this kind to assess the more general applicability of our results. Together, this sets the stage for our contributions.

Based on earlier work on quality of conceptual models, an improved framework for model and language quality based on social construction theory is created and used for evaluation of approaches to support computerized information systems applying conceptual modeling.

A general framework for computerized information systems support in organizations based on the perceived current situation and our philosophical outlook is presented, and the suggestions are further substantiated by indicating how conceptual modeling languages can be used within the framework in more detail.

The baseline of the work on conceptual modeling languages is the structural, functional, and behavioral based approach PPP which has been developed in the Information Systems Group at NTH, and the structural, functional, and rule-based approach which has been developed in the ESPRIT project Tempora. This choice of modeling languages is partly pragmatic, since we have a long time experience with these frameworks in our research group, and also have available tools supporting main parts of these approaches. The primary assessment of these approaches using the quality framework also indicates that although certain deficiencies can be identified, these approaches have a potential, with the right tool and method support, to be used to create conceptual models of high quality. The evaluation uncovered a limited representation of both social and technical actors in these approaches, in addition to the lacking support of the rule perspective in PPP. Suggestions to address these weaknesses are given based on that the languages should be useful in the overall framework, and it is illustrated how the complete framework can be used more concretely in the methodological framework.

The best way to investigate if a suggested approach works, is to use it. This presumes in our case that a tool environment exists. Unfortunately, the current implementation of PPP was at such an early state that it was not possible for us to extend this with our suggestions within the time set for the thesis. A massive effort is needed to implement a full-fledged CASE tool separately, and this goes beyond what can be achieved within the scope of a Ph.D. thesis. Therefore, the validation of the suggested approach of this thesis is based on the use of medium-sized case-studies. The extensions of the conceptual framework are also evaluated based on the quality framework, and by comparisons with other approaches described in the literature.

Major Contributions

The major results of this thesis are deemed to be the following:

- We have performed a survey investigation on development and maintenance practice in Norwegian organizations giving partly novel results, especially on the distinction between functional development and functional maintenance.
- We have combined social construction theory with conceptual modeling. Although also other approaches based on social construction theory have applied traditional conceptual modeling languages such as DFD and ER, it has to our knowledge not been done with a more comprehensive framework for conceptual modeling which support many modeling perspectives in an integrated manner.
- We have extended an existing framework for the understanding of quality of conceptual models and conceptual modeling languages inspired by social construction theory.
- We have proposed a methodological framework combining development and maintenance having support for not only the single application systems, but the whole application system portfolio. Other important aspects of the framework are social construction theory, reuse, and conceptual modeling.
- An actor-modeling language for the representation of both social and technical actors has been developed and integrated in the conceptual framework. Support for rule-modeling has also been extended based on results from Tempora, with improved possibility to cover deontic modalities both within and between rules.

Outline of the Thesis

The thesis is structured in 12 chapters as follows:

Chapter 1 presents the basic philosophical assumptions of the thesis, namely the view that reality is socially constructed and undergoes continuous change. How this relates to computerized information systems support in organizations is briefly outlined. Chapter 2 contains a definition of the main part of the terminology used in the thesis. The terminology is influenced by our constructivistic outlook. Different changes in a computerized information system as discussed in Chapter 1 are covered by a typology of tasks in development and maintenance. A more comprehensive definition of the terminology is given in Appendix H. Problems in connection to computerized information system support in organizations are then discussed in the light of the philosophical outlook presented in Chapter 1.

Chapter 3 contains an overview of some of the results from our survey investigation on development and maintenance of computerized information systems, including comparisons with the results from similar investigations having been performed in the past on this area by other researchers. A further description of the conduct of the investigation is found in Appendix F.

Chapter 4 contains an overview of methodologies for computerized information system support in organizations. Five areas are used for classification:

- "Weltanschauung": Based on the discussion in Chapter 1, a constructivistic "Weltanschauung" is proposed.
- Coverage in process: Based on the discussion in Chapter 1 and results presented in Chapter 3, we recommend a methodology that look upon development, use, and maintenance in an integrated manner.

- Coverage in product: Based on results from Chapter 3 we propose a methodology supporting the whole portfolio of application systems in an organization, and not only individual systems.
- Reuse of product and process: Primarily motivated by the rapid changes that faces today's organizations as discussed in Chapter 1.
- Conceptual modeling: The use of conceptual modeling is motivated based on how it can support the four other aspects of the classification.

Major trends in methodologies for computerized information systems support in organizations are then outlined using the above classification. In the end of the chapter, the need for flexibility in a methodology is discussed based on the material presented in the three first chapters.

In Chapter 5 conceptual modeling is discussed in more detail in the light of social construction, and different perspectives to modeling are presented. In Chapter 6 a framework for the understanding of quality in conceptual modeling is presented. Chapter 7 presents some approaches to conceptual modeling that applies a combination of languages with different modeling perspectives in more detail. This also includes presentations of PPP and Tempora. All approaches are classified and briefly evaluated according to the framework presented in Chapter 6.

The overall methodological framework is presented in Chapter 8 and is based on the discussion from Chapter 5. It also contains a description of stakeholder participation based on material from the first three chapters. Extensions of the conceptual framework of PPP are given in Chapter 9. The extensions are made to better support the four main aspects of the methodology as outlined in Chapter 4, taking into account existing work within these perspectives as discussed in Chapter 5 and the discussion of language quality from Chapter 6. In Chapter 10 we indicate how the combined conceptual framework as described in Chapter 7 and Chapter 9 can be used within the methodological framework presented in Chapter 8. Aspects from the framework for conceptual modeling quality are used and extended to indicate its use in practice. The overall introduction of the methodological framework is discussed on the background of the material presented in Chapter 3.

An evaluation of the proposals from Chapters 8- 10 is given based on the classification given in Chapter 4 and the quality framework given in Chapter 6. Chapter 12 summarizes the contributions of the thesis as presented in Chapter 3, Chapter 6, and Chapters 8- 10, in addition to pointing to areas for further work which have been suggested throughout the theses.

A comprehensive description of the case-studies used throughout the thesis is found in Appendix A. The cases include an overall model of the information systems support at our institute, an application system supporting the arrangement of a professional conference, the Sweden Post case-study used in Tempora and the library case also originally used in Tempora. An overview of mathematical symbols used in the thesis is given in Appendix G.

About the Work

During the working period, the author has collaborated with a number of people. To make sure that the thesis cites the sources for anything that is not original with the author, we adopt the following conventions:

- Work done by others: The source of the contribution is cited.
- Work done in collaboration with others where it is difficult to distinguish who has contributed what: It will be explicitly stated in the text that the work reported is done together with other people.

- Work done by project students or M.Sc students supervised by the author: The particular report is cited.

Additional material should be regarded as the contributions of the author, although obviously inspired by the work of other researchers, especially the work surveyed in the state-of-the-art chapters.

Acknowledgments

I would like to thank my supervisor Professor Arne Sølvberg and the rest of the Institute for competent support and interesting discussions during the work. Especially I would like to mention Anne Helga Seltveit, Odd Ivar Lindland, Guttorm Sindre, Geir Willumsen, Jon Atle Gulla, Harald Rønneberg, Sigurd Thunem, Geir Magne Høydalsvik, Eirik Tryggeseth, Reidar Conradi, Steinar Carlsen, and Rudolf Andersen. Also thanks to the students I have supervised as part of my Ph.D-studies, especially Odd Noreide, Anne Grethe Eilertsen, Babak Amin Farshchian, Mette Rønning Raabel, and the group developing the conference system used in the main case-study, especially Kurt Petter Humstad and Vidar Larsen.

Of the persons outside the institute which have influenced my work, I would particularly like to mention Peter McBrien, Richard Owens, Reidar Gjersvik, Arne Henne, and all referees of papers being submitted to different conferences and journals. Also thanks to all the respondents and piloters of the survey investigation.

Also thanks to Andersen Consulting ANS, who offered me leave of absence to do the Ph.D. studies. Especially I would like to thank Joe Taraldrud, Lars Henriksen, Helge Clem, Pål Fevang, Nils Øveraas, and Paul Arne Kristensen.

Finally, thanks to Birgit, who more than anyone else has looked forward to the completion of this thesis.

NTH Trondheim, December 21, 1995

John Krogstie

Contents

1 Philosophical Outlook **1**
- 1.1 The Social Construction of Reality . . . 1
- 1.2 A World in Change . . . 3
- 1.3 Chapter Summary . . . 6

2 Computerized Information Systems **7**
- 2.1 Terminology . . . 7
  - 2.1.1 Assumptions underlying the Terminology . . . 8
  - 2.1.2 Phenomena and States . . . 8
  - 2.1.3 Data, Information, and Knowledge . . . 9
  - 2.1.4 Language and Models . . . 9
  - 2.1.5 Actors and Activities . . . 10
  - 2.1.6 Systems . . . 11
  - 2.1.7 Methodology . . . 12
- 2.2 Problems with Computerized Information Systems . . . 14
  - 2.2.1 Why is there a Problem? . . . 14
- 2.3 Chapter Summary . . . 16

3 Development and Maintenance of the COIS **17**
- 3.1 Introduction to the Survey Investigation . . . 17
- 3.2 Computerized Information System Support in Organizations . . . 18
  - 3.2.1 Distribution of Effort on Development and Maintenance . . . 19
  - 3.2.2 Organization of Development and Maintenance . . . 22
  - 3.2.3 Methods and Tools . . . 23
  - 3.2.4 Portfolio Analysis . . . 25
  - 3.2.5 Replacement Systems . . . 29
  - 3.2.6 Perceived Problems in Maintenance . . . 30
- 3.3 Chapter Summary . . . 32

4 Methodologies for Computerized Information Systems Support **35**
- 4.1 Classification of Methodological Frameworks . . . 35
- 4.2 A Survey of Methodological Frameworks . . . 41
  - 4.2.1 The Waterfall Model . . . 41
  - 4.2.2 Alternatives and Extensions of the Waterfall Model . . . 43
  - 4.2.3 Method/1 . . . 47
  - 4.2.4 The Spiral Model . . . 48
  - 4.2.5 The Hierarchical Spiral Model . . . 50

4.2.6 Object-oriented Approaches and the Fountain Model 52
4.2.7 REBOOT 54
4.2.8 CONFORM 55
4.2.9 Maintenance as Reuse-oriented Development 57
4.2.10 Multiview 59
4.2.11 STEPS 61
4.3 Chapter Summary 62

5 Conceptual Modeling 65
5.1 Conceptual Modeling as Social Construction 66
5.2 Overview of Languages for Conceptual Modeling 68
5.2.1 The Structural Perspective 69
5.2.2 The Functional Perspective 71
5.2.3 The Behavioral Perspective 73
5.2.4 The Rule Perspective 76
5.2.5 The Object Perspective 81
5.2.6 The Communication Perspective 82
5.2.7 The Actor and Role Perspective 86
5.3 On Conceptual Relativism 90
5.4 Chapter Summary 92

6 Quality of Conceptual Models 93
6.1 Overview and Evaluation of Existing Frameworks 93
6.1.1 Lindland/Sindre/Sølvberg's Framework 93
6.1.2 Pohl's Framework 94
6.1.3 The Use of the Semiotic Ladder in FRISCO 95
6.1.4 Overall Comparison 96
6.1.5 Critique of the Framework of Lindland et al. 97
6.2 A Framework for Quality of Conceptual Models 97
6.2.1 The Term 'Statement' 101
6.2.2 Physical Quality 102
6.2.3 Syntactic Quality 102
6.2.4 Semantic Quality 103
6.2.5 Perceived Semantic Quality 106
6.2.6 Pragmatic Quality 107
6.2.7 Social Quality 111
6.2.8 Internalization 113
6.2.9 Knowledge Quality 113
6.2.10 Language Quality 114
6.3 Chapter Summary 119

7 Multi-perspective Approaches to Conceptual Modeling 121
7.1 OMT 121
7.1.1 Languages for Conceptual Modeling 121
7.1.2 Methodology 123
7.1.3 Evaluation 126
7.2 Tempora 127

7.2.1 Languages for Conceptual Modeling . . . 127
7.2.2 Methodology . . . 132
7.2.3 Evaluation . . . 135
7.3 SAMPO . . . 137
7.3.1 Languages for Conceptual Modeling . . . 137
7.3.2 Methodology . . . 144
7.3.3 Evaluation . . . 144
7.4 PPP . . . 145
7.4.1 Languages for Conceptual Modeling . . . 145
7.4.2 Methodology . . . 153
7.4.3 Evaluation . . . 155
7.5 Chapter Summary . . . 157

8 Methodological Framework 159
8.1 System Devtenance . . . 159
8.2 Application System Life Cycle . . . 161
8.3 Stakeholder Participation in System Devtenance . . . 165
8.3.1 Background on Participation . . . 165
8.3.2 Principles of Stakeholder Participation . . . 167
8.4 Chapter Summary . . . 169

9 Conceptual Framework 171
9.1 Extended Actor Modeling Language . . . 172
9.1.1 Modeling Language . . . 172
9.2 DRL: Deontic Rule Language . . . 181
9.2.1 Intra-rule Deontic Operators . . . 183
9.2.2 Inter-rule Deontic Operators . . . 184
9.2.3 Speech Acts and Deontic Rules . . . 188
9.3 Integration with PPP . . . 190
9.3.1 Extension of the Meta-Model . . . 191
9.3.2 Extension of Prototype Generation . . . 195
9.3.3 Extension of Consistency Checking . . . 202
9.3.4 Driving Questions . . . 203
9.4 Chapter Summary . . . 206

10 Conceptual Modeling in Systems Devtenance 209
10.1 Introducing System Devtenance in an Organization . . . 210
10.2 Conceptual Modeling in Systems Devtenance . . . 214
10.2.1 Process Heuristics in Conceptual Modeling . . . 215
10.2.2 Devtenance with no Preliminary Conceptual Descriptions . . . 219
10.2.3 Conceptual Models in Usage Support . . . 240
10.2.4 Conceptual Modeling Based on Existing Conceptual Descriptions . . . 243
10.3 Chapter Summary . . . 245

11 Evaluation **247**
11.1 Classification of Methodological Framework . . . 247
11.2 Evaluation of Language Quality . . . 248
11.2.1 Domain Appropriateness . . . 248
11.2.2 Participant Knowledge Appropriateness . . . 253
11.2.3 Participant Interpretation Enhancement . . . 253
11.2.4 Technical Actor Interpretation Enhancement . . . 254
11.3 Potential for Creating Models of High Quality . . . 254
11.4 Chapter Summary . . . 255

12 Concluding Remarks and Future Directions **257**
12.1 Major Contributions . . . 257
12.2 Limitations and Future Directions . . . 258
12.3 Model Integration in System Devtenance . . . 259
12.3.1 Intra-project Model Integration . . . 259
12.3.2 Inter-project Model Integration . . . 260
12.3.3 Inter-organizational Model Integration . . . 262
12.3.4 Outline of an Approach to Model Integration . . . 262
12.4 Chapter Summary . . . 263

A Case-studies **265**
A.1 CIS-support at IDT . . . 265
A.1.1 IDT-case . . . 266
A.1.2 The Conference System . . . 302
A.2 Tempora-cases . . . 397
A.2.1 Sweden Post . . . 398
A.2.2 The Library Case . . . 399

B PPP Repository Structure **401**

C DRL Syntax **403**

D Integration of the Conceptual Extensions and PPP Modeling Techniques **407**
D.1 General Execution . . . 407
D.1.1 Incorporating Time . . . 411
D.1.2 Actors and Roles . . . 412
D.1.3 Processes and Rules . . . 412
D.1.4 Deontic Constraints . . . 413
D.1.5 Timers . . . 414
D.2 Explanation Generation . . . 415
D.2.1 Actors and Roles . . . 418
D.2.2 Processes and Rules . . . 418
D.2.3 Rule Hierarchies . . . 420
D.2.4 Deontic Constraints . . . 422
D.2.5 Timers . . . 422
D.3 Filtering . . . 423
D.3.1 ONER Filters . . . 425
D.3.2 PPM Filters . . . 425

D.3.3 Rule Filters . . . 426
D.3.4 Rule Relationship Filters . . . 427
D.3.5 Actor Model Filters . . . 427
D.3.6 Filters Combining Models in Several Languages . . . 428
D.4 Additional Tool Support . . . 429

E Statements in PPP **435**
E.1 Language for the Representation of Statements . . . 435
E.2 ONER Models . . . 436
E.3 PPM . . . 438
E.4 Actor Models . . . 440
E.5 Rules . . . 440
E.6 Metalevel Statements . . . 441

F Description of Survey **443**
F.1 Respondents . . . 444
F.2 Analysis . . . 444
F.3 Critique of the Survey Technique . . . 445
F.4 Statistical Methods . . . 445
F.4.1 Hypothesis Testing . . . 446
F.4.2 The Correlation Coefficient . . . 447
F.5 Questionnaire . . . 448

G Mathematical Symbols **469**

H Terminology **471**
H.1 Time . . . 471
H.2 Phenomena . . . 472
H.3 State and Rules . . . 473
H.4 Data, Information, and Knowledge . . . 475
H.5 Language and Models . . . 476
H.6 Actors and Activities . . . 478
H.7 Systems . . . 480
H.8 Social Construction . . . 482
H.9 Methodology . . . 482
H.10 Abbreviations . . . 485

Bibliography **495**

List of Figures

1.1 Social construction in an organization . . . 3
1.2 A typology of changes . . . 4

2.1 Actor class hierarchy . . . 10
2.2 Relationship between functional and traditional development and maintenance . . 14

4.1 The waterfall model of system development (From [28]) . . . 42
4.2 Iterative vs. throwaway prototyping (From [44]) . . . 45
4.3 The transformational and operational life cycle models . . . 46
4.4 Method/1 for customized systems . . . 48
4.5 The spiral model (From [29]) . . . 49
4.6 The hierarchical spiral model (From [139]) . . . 51
4.7 The Fountain model (From [129]) . . . 53
4.8 The software maintenance model . . . 56
4.9 Maintenance process models . . . 58
4.10 The phases and interdependencies in Multiview (From [17]) . . . 60
4.11 The STEPS methodology (From [90]) . . . 61

5.1 Example of an ER-model . . . 69
5.2 Example of a GSM model . . . 70
5.3 A conceptual graph linked to a semantic network (From [294]) . . . 71
5.4 Symbols in the DFD language . . . 71
5.5 Symbols in the transformation schema language . . . 72
5.6 Symbols in the real-world modeling language . . . 74
5.7 Symbols in the state transition modeling language . . . 74
5.8 Example of a state transition model . . . 75
5.9 Decomposition mechanisms in Statecharts . . . 75
5.10 Dynamic expressiveness of Petri-nets . . . 76
5.11 Example of a goal hierarchy (From [297]) . . . 79
5.12 Example of a goal-graph (From [55]) . . . 80
5.13 Example of viewpoint hierarchies . . . 81
5.14 General object model(From [332]) . . . 82
5.15 Conversation for action (From [337]) . . . 83
5.16 Main phases of action workflow . . . 84
5.17 Comparing communicative action in Habermas and Searle (From [72]) . . . 85
5.18 The symbols of the ABC-language (From [71]) . . . 86
5.19 Example of an ALBERT model (From [78]) . . . 87
5.20 Example of an actor dependency model (From [344]) . . . 89

5.21 Symbols in agents-role-position modeling language (From [343]) 90
5.22 Symbols in the OORASS role interaction language 90

6.1 The framework for model quality by Lindland et al. (From [199]) 94
6.2 Pohl's framework for the requirement specification process (From [261]) 95
6.3 Extended framework for discussing quality of conceptual models 98
6.4 Statements in a simple ER-diagram 101
6.5 Example of syntactic invalidity 103
6.6 Example of syntactic incompleteness 103
6.7 Example of semantic invalidity 104
6.8 Example of a syntactically complete delete 105
6.9 Example on poor aesthetics 109
6.10 Example of a language filter 110
6.11 Example of a model filter 110
6.12 The example written in the GSM language 111

7.1 Symbols in the OMT object modeling language 123
7.2 Example of an OMT object model 124
7.3 Symbols in the OMT dynamic modeling language 125
7.4 Symbols in the ERT languages 128
7.5 Example of an ERT model 129
7.6 Symbols in the PID language 129
7.7 Example of a PID model 130
7.8 Relationship between the PID and ERL languages (from [179]) 131
7.9 The main phases of requirements specification in Tempora 133
7.10 From IS-model to CIS-model in Tempora 134
7.11 Symbols in the discourse modeling language (From [15]) 138
7.12 Example of a discourse model (From [15]) 139
7.13 Detailed model of invitation of speaker(From [15]) 140
7.14 Symbols in the conversation modeling language(From [15]) 141
7.15 Example of a conversation model(From [15]) 143
7.16 Symbols in the ONER language 146
7.17 Example of an ONER-model 148
7.18 Symbols in the PPM language 149
7.19 Example of a PLD model 151
7.20 Example on an actor model 151
7.21 Symbols in the extensions to Statecharts used in UDD 153
7.22 CIS-development strategy in PPP 153

8.1 A framework for system devtenance 160
8.2 Actor model describing a project organization 161
8.3 Nested transactions in system devtenance 162
8.4 System devtenance life-cycle 164
8.5 Scale of influence and power in participation (From [128]) 165
8.6 Co-generative learning in systems devtenance 167

9.1 Basis for the actor modeling language 173
9.2 Overall organizational model of IDT 176

9.3 PPM describing creation and distribution of Call for Papers . . . 178
9.4 Simple use of illocutionary acts . . . 179
9.5 Illocutionary acts in PPM . . . 179
9.6 Conversation for action in PPM . . . 180
9.7 Invitation of speakers in PPP . . . 181
9.8 Detailed invitation of speaker in PPM . . . 182
9.9 Mechanism in rule-hierarchies . . . 186
9.10 Example of a rule hierarchy . . . 188
9.11 Extensions of the meta-meta-model in PPP . . . 191
9.12 Incorporating the use of rules and actors in the PPP meta-model . . . 192
9.13 Meta-model based on the actor-modeling language . . . 194
9.14 Updated meta-model for ONER . . . 195
9.15 Relationship between rule-hierarchies and IBIS . . . 196
9.16 Over-ruling mechanisms based on the actor model . . . 201

10.1 Overall organizational model of IS-group . . . 211
10.2 Example of technical actor integration diagram . . . 212
10.3 Example of social actor support . . . 213
10.4 Example of technical actor support . . . 214
10.5 The SPEC-cycle for modeling . . . 216
10.6 Overall actor model of ISDO95 . . . 221
10.7 Actor model of project-participants . . . 222
10.8 Rules based on CATWOE analysis . . . 225
10.9 PPM describing CFP-distribution at the actor-level . . . 227
10.10 PPM describing reception and distribution of papers based on $\mathcal{K}_2$ . . . 229
10.11 PPM describing distribution of papers based on $\mathcal{K}_1$ . . . 230
10.12 PPM describing the paper handling . . . 232
10.13 PPM describing address-registration . . . 233
10.14 PPM describing specifications of reports . . . 234
10.15 Goal-hierarchy extended from CATWOE analysis . . . 237
10.16 IBIS-network on the issue of system platform . . . 238
10.17 Pruned goal-hierarchy after argumentation process . . . 239
10.18 Actor model of the C4 application system . . . 240
10.19 Conceptual modeling in a devtenance project . . . 241
10.20 PPM describing invitation of discussants . . . 244
10.21 Conceptual modeling in a follow-up project . . . 245

11.1 PPM for library example . . . 252

12.1 ONER-model for the traditional IFIP-case (From [340]) . . . 261

A.1 Actor model describing IDT . . . 266
A.2 ONER-model describing the static aspects of formal roles at IDT . . . 268
A.3 Actor model describing aspects of the role 'professor' . . . 271
A.4 Actor model describing actors with multiple simultaneous roles . . . 272
A.5 Actor model describing the education committee . . . 273
A.6 PPM describing the working pattern of the education committee . . . 275
A.7 Actor model describing the IS group . . . 278

A.8 Actor model describing social support relationships involving the IS-group . . . 281
A.9 Actor model describing support-relations between technical and social actors . . 287
A.10 Actor model describing all technical actors supporting a selected individual actor 288
A.11 Actor model describing roles of data-stores 292
A.12 PPM describing basic capabilities of a DBMS 293
A.13 Actor model describing a communication pattern between social and technical actors 294
A.14 Actor model describing the parts of the bibutil-actor 296
A.15 Indicating control-flow in parts of bibutil 297
A.16 PPM describing parts of bibutil 299
A.17 Actor model describing actors and roles in the project-group 304
A.18 Actor model describing the conference organization 307
A.19 Actor model describing the group-project 311
A.20 PPM describing creation and distribution of CFP 313
A.21 PPM describing reception and distributions of papers 314
A.22 PPM describing reception of CRC 316
A.23 PPM describing distribution of papers to reviewers 318
A.24 PPM describing selection of papers 320
A.25 Actor model describing interactions in connection with conference organization . 323
A.26 Overall reconciled PPM 326
A.27 Speech acts in the paper-process 329
A.28 ONER-model for the conference system 333
A.29 PPM describing new creation and distribution of CFP 338
A.30 PPM describing new reception of papers 340
A.31 PPM describing new distribution of paper 343
A.32 PPM describing new reception of reviews 346
A.33 PPM describing new reminding of reviews 350
A.34 PPM describing new selection of papers 352
A.35 PPM describing new notification of results 353
A.36 PPM describing create review-reports 355
A.37 PPM describing new reception of CRC 357
A.38 PPM describing new reminding on CRC 359
A.39 PPM describing new preparation of manuscript 361
A.40 PPM describing registration and update of addresses 363
A.41 Scenario to ONER-diagram relevant to the address-register 364
A.42 PLD for Process 8.1: Receive address 364
A.43 PLD for Process 8.2: Check if already exist 365
A.44 PLD for Process 8.3: Send confirmation 366
A.45 PLD for Process 8.4: Delete address 367
A.46 Actor model describing the C4 application system 370
A.47 Example of rule hierarchy built bottom-up 398

B.1 The PPP repository structure 401

D.1 Data modeling language for meta-modeling (from [335]) 408
D.2 Meta-model for execution in PPP 408
D.3 Execution of deontic rules 412

D.4 Steps and representations in explanation generation 416
D.5 Domain structure model 416
D.6 Representation of actors and roles 418
D.7 Integration of process and rule in PPP 418
D.8 Explanation generation based on rule-relationships 420
D.9 Scenario for explanation 421
D.10 Explanation structure 421
D.11 Example of tool-layout 429

H.1 Time and duration 472

List of Tables

3.1 Descriptive data for size-measures . . . 18
3.2 Tests for normality of size measures . . . 19
3.3 Distribution of work effort . . . 20
3.4 Comparing the work distribution in various investigations . . . 21
3.5 Tests for normality of maintenance figures . . . 21
3.6 Testing of equality of work distribution . . . 22
3.7 Testing relationships between functional and traditional maintenance measures . 22
3.8 Maintenance and size vs. use of overall methodology . . . 24
3.9 Use of organizational controls . . . 25
3.10 Maintenance and size vs. use of CASE . . . 26
3.11 Use of CASE-tools . . . 26
3.12 Age distribution of major systems . . . 27
3.13 Development method vs. size . . . 28
3.14 Use of programming languages . . . 29
3.15 Reasons for replacement . . . 30
3.16 Problem areas in maintenance . . . 31
3.17 Assessed problems in the Chapin-investigation [49] . . . 32
3.18 Assessed problems in the Dekleva-investigation [67] . . . 33

4.1 The REBOOT key reuse areas and factors . . . 54
4.2 Classification of Methodologies . . . 63

5.1 A data flow diagram taxonomy of real-world dynamics . . . 73

6.1 Framework for model quality . . . 100
6.2 A taxonomy of graph aesthetics (From [302]) . . . 109
6.3 A taxonomy of constraints for graph layout (From [302]) . . . 110

8.1 Links between users and developers in CIS-support . . . 166

D.1 Relationships in meta-model for general execution . . . 410
D.2 Relationships in model for explanation generation . . . 417

Chapter 1

Philosophical Outlook

"Homo Mensura"	*"Panta Rei"*
Protagoras	*Heraclit*

1.1 The Social Construction of Reality

According to Guba and Lincoln [111] one reality that is measurable and essentially the same for all is generally believed to exist. This belief system, which is often called the conventional or objectivistic belief system, has the following characteristics:

- Ontology (what is there that can be known): The ontology in the belief system is realistic, asserting that there exist a single reality that is independent of every observer's interest in it and which operates according to immutable natural laws. Truth is defined as that set of statements that is isomorphic to reality.
- Epistemology (what is the relationship between the knower and the known): The epistemology of this belief system is one of dualistic objectivism, asserting that it is possible, indeed mandatory for an observer to exteriorize the phenomena studied, remaining detached and distant from it and excluding any value considerations from influencing it.
- Methodology (what are the ways of achieving knowledge): The methodology of this belief system is one of interventionism, stripping context of its contaminating influences so that the inquiry can converge on truth and explain the things studied as they really are and really work, leading to the capability to predict and to control.

The conventional belief system falls short of being able to deal with emergent conceptual and empirical formulations and results from a variety of fields [194]. Some well-known examples are:

- Gödels Incompleteness Theorem, which states that no axiomatic system of mathematics is able to provide information about both the completeness and consistency of that system.
- Heisenbergs's Uncertainty Principle, which assert that the position and momentum of an electron cannot both be determined exactly, because the action of the observer in making either measurement inevitably alters the other.
- Bell's theorem, which asserts that no theory compatible with quantum theory can require spatially separated events to be independent.

Based on all this, both determinism and reductionism, which are consequences of the conventional belief system, seems unfounded.

To counter these results, one has had to perform a paradigmatic revolution in many areas, replacing conventional beliefs with a constructivistic belief system having the following characteristics:

- Ontology: The ontology in this belief system is one of relativism, asserting that there exist multiple socially constructed realities ungoverned by any natural laws, causal and otherwise. "Truth" is defined as the best informed and most sophisticated construction on which there is agreement.
- Epistemology: The epistemology of this belief system is subjectivistic, asserting that the inquirer and the inquired-into are interlocked in such a way that the findings of an investigation are the literal creation of the inquiry process.
- Methodology: Hermeneutics, involving a continuing dialectic of iteration, analysis, critique, reiteration, reanalysis, and so on leading to the emergence of a joint construction of the case among all the inquirers and respondents is looked upon as the most appropriate methodology in this belief system.

Most of the distinguished features of the constructivistic paradigm have emerged from fields such as physics and chemistry, customarily perceived as the "hard" natural sciences. The issue of what reality is, is very much up for grabs even in areas like physics in which one should suppose such questions to have been settled long time ago. The argument for the new paradigm can be made even more persuasively when the phenomena being studied involve human beings, i.e. in the "soft", social sciences. A great deal of the theoretical discussion in the social sciences is today dedicated to analyzing constructivism and its consequences [59]. Influential empirical work in such areas as sociology of science and social problems are guided by the idea of reality construction. This idea has been the central topic for the philosophical debate in the last two decades, approached in different fashions by French, American, and German philosophers.

Whereas many different approaches to constructivistic thinking have appeared, probably the most influential one is the one of Berger and Luckmann [23], and their insights on this matter will also be used as our starting point. Their view of social construction of reality is based on Husserl's phenomenology. Whereas Husserl was primarily a philosopher, Schutz took phenomenology into the social sciences. From there on it branched into two directions: Ethnomethodology, primarily developed by Garfinkel and the social constructivism of Berger and Luckmann. Whereas ethnomethodology is focused on questioning what individuals take as given in different cultures, Berger and Luckman developed their approach to investigate *how* these presumptions are constructed.

According to Gjersvik [102], organizations are realities socially constructed through the joint actions of the social actors in the organization. This process as illustrated in Figure 1.1 and is describe below.

An organization consists of individuals who see the world in a way specific to them because they have different experiences from work and from other arenas. The *local reality* is the way the individual perceives the world that he or she acts in. The local reality is just the way the world is for the individual, it is the everyday perceived reality of the individual social actor. Some of this may be made explicit and talked about. However, a lot of what we know is *tacit*. When the social actors of an organization act, they *externalize* their local reality. The most important ways social actors externalize their local reality, are by speaking and constructing languages, artifacts, and institution. What they do, is to construct *organizational reality*: To make something that other actors have to relate to as being part of the organization. This organizational reality may consist of different things, such as institutions, language, artifacts, and technology. Finally, *internalization* is the process of making sense out of the actions, institutions, artifacts etc. in the organization,

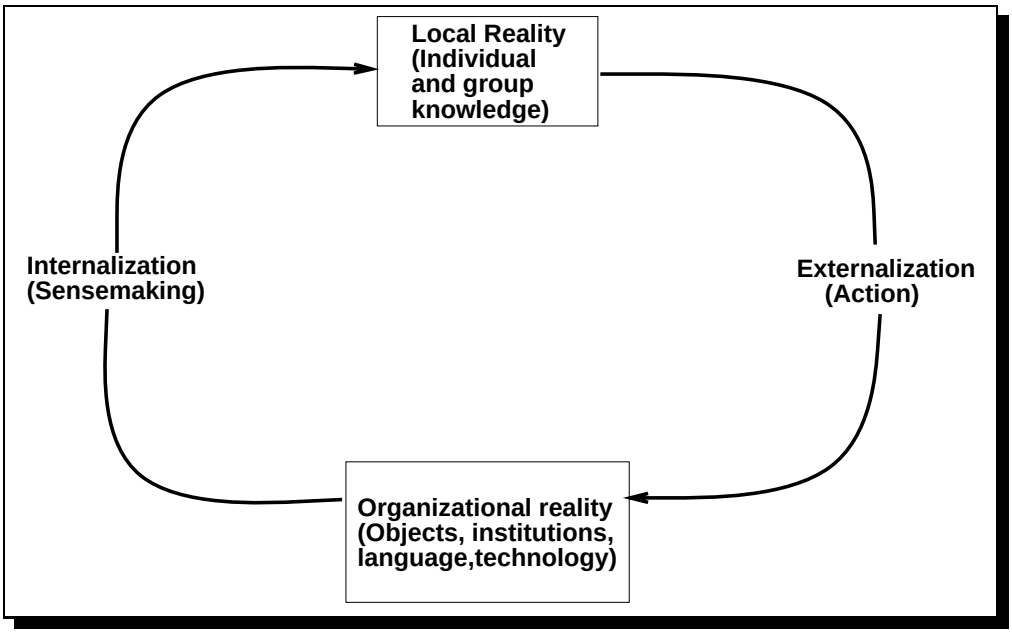

Figure 1.1: Social construction in an organization

and making this organizational reality part of the individual local reality. This do not mean that the processes of externalization and internalization are going on in a strict sequence, both externalization and internalization might be performed simultaneously. It neither means that it is only the organizational reality in one organization that is internalized by an individual, also other externalizations influence the local reality of an individual.

The change of the computerized information system support in an organization through for instance developing new application systems can also be looked upon as a process of social construction, a view that has received increasing interest in the information systems community in the last years [91, 195, 285, 337] and this outlook will also be used in this thesis, especially concentrated on the creation and maintenance of conceptual models in connection with improving the computerized information system support of an organization. This do not mean that we are ignorant of the more technical aspects of computerized information systems support, constructed realities are often related to, and equally often inseparable from, tangible phenomena.

1.2 A World in Change

Organizations are continuously under the pressure to change. The change-inducing forces can be divided into several categories. In organizational theory, a distinction is often made between the external and internal environment of the organization, where the external environment again is divided into the general environment and the task environment [135]:

- The task environment contains those components of the external environment that interact with the organization on a regular basis and/or have a major impact on it. Typical components of a task environment include customers, competitors, suppliers, and specific governmental agencies. Changes in the task environment such as increased competition, changes in consumer tastes and product preferences, and changing cost of resources have a direct influence on the organization.

- The general environment contains those components of the external environment that have more indirect effect on the organization. Changes in this environment can be of different kinds:
  - Technological, with new production or information processing technology.
  - Economic, such as globalisation of the economy and a shift to a service economy.
  - Social, political, or demographic exemplified with an aging population, changes in family structure, changing values towards work, and changes in governmental legislation.

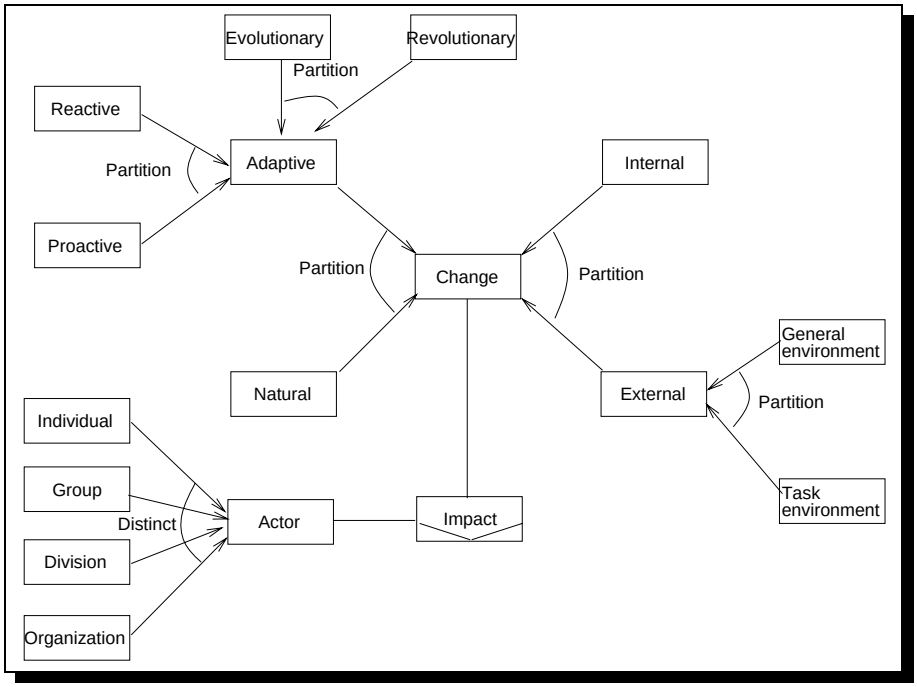

Figure 1.2: A typology of changes

Changes performed in these environments will be termed external relative to the organization whereas changes taking place within the boundary of the organization are termed internal. A typology of changes is given in Figure 1.2 and described below. Both internal and external changes can be divided into two groups, so-called natural changes, which are changes that occur in organizations, whether purposely performed by organizational actors or not, and adaptive changes which are performed consciously to change the current state of affairs, adapting to the internal or external threats and possibilities that the organization is faced with [233]. An example of natural, internal changes are changes due to the increasing size of an organization. A natural, external change can be understood using a population ecology model which is inspired by natural selection theory of biology [233]. For us, the adaptive changes being performed to address the perceived internal or external discrepancies are of primary interest. Adaptive changes can either be reactive, trying to respond to the changes either in the external or internal environment by repositioning the organization at the same perceived relative position as previously held. Adaptive changes might also be pro-active, setting a new goal for the relative position of the company, and striving to achieve this new goal. Reactive changes are usually performed when one are faced with perceived problems, whereas pro-active changes are often done when sensing an opportunity. Usually, the changes induced in an organization will have both reactive and pro-active elements

over time. The results of the induced changes might be as expected, or the changes have unintended consequences, inducing the need for performing additional change. Performing changes in the organization will often also induce a need for actors in the task environment to perform reactive changes, and at times even influence actors in the organization's general environment to change. Both reactive and pro-active adaptive changes can be classified as being either evolutionary or revolutionary. An evolutionary adaptive change is based on the existing assumptions about how the organization is supposed to function, whereas a revolutionary change is a result of first reevaluating the existing assumptions behind the activities of the organization, and then perform the changes. Finally, we can mention that the impact of changes in an organization can have varying scope, being on an individual, group, divisional, organizational, or inter-organizational level respectively. The scope of a change includes all persons who perceive or is perceived by other persons to lose or gain from a change taking place. These persons are the stakeholders of the activity resulting in a change.

When we look into the changes being done on the computerized information system support of an organizations, need for change manifest itself in many ways. Below, the perceived situation is described first, and then the resulting change on the computerized information system.

- A discrepancy between what an application system does and what it is supposed do to, i.e. a fault, is discovered. The change will be the removal of the fault. This will be an adaptive, reactive, and evolutionary change based on the existing situation in the internal environment. The scope of the change is all persons using the parts of the application system which are changed.
- Several application systems exist within the organization, which by themselves function as expected, but which are not consistent regarding for instance the interpretation of commonly used data. The need for change is uncovered when the systems have to inter-operate. For the part that have to change, this will be an adaptive, reactive, and partly revolutionary change based on the existing situation in the internal environment of the organization, the scope being the department where the system that is changed is used.
- On a high organizational level, a wish to standardize on some solution for a given area can induce a need for changing systems not adhering to this standard. The standardization can be on supportive software or hardware or can be a standardization of a more methodological nature, e.g. how computerized information systems are developed and maintained within the organization. This is an adaptive, reactive and revolutionary change based on reactive or pro-active changes in the internal environment.
- Inter-organizational cooperation using for instance EDI might induce changes of the same kind across organizational borders. The change will be adaptive, reactive, and revolutionary, based on changes in the external task environment, and will influence both the internal and the external task environment.
- A change in the general environment, especially the technological environment, for instance that the computer manufacturer stops the support of certain hardware being used in the organization, induce the need for the adaptive, reactive, evolutionary change of moving the application systems to a new hardware platform.
- A natural internal change happens, for instance that the organization grows. This can result in worse performance of the application system, and thus a need for an adaptive, reactive, evolutionary change. On the other hand, when such problems are not yet present, but expected in the future, we might have an adaptive, pro-active, evolutionary change.
- Another interesting internal change is that caused by the continuous externalization and internalization on different areas, i.e. the actors of the organizations learns. This can result

in the discovery of new needs for computerized information systems support, and thus in adaptive, reactive or pro-active revolutionary changes.
- Natural changes in the task environment e.g. that all organizations in the same business decides to offer the same services, because their competitors do can often result in adaptive, reactive, evolutionary changes.
- Changes in the external environment combined with increased knowledge about what computerized information systems can be used for can result in adaptive, pro-active, and revolutionary changes e.g. through so-called BPR.
- Finally, it can be mentioned that due to the continuous changes of computerized information systems, they will be going through a natural change of increasing complexity if not preventive actions are performed to avoid it [189].

Even if there seems to be a general agreement on that most modern western organizations are living in a turbulent, ever-changing environment [233], there is disagreement about the underlying reasons for this. We will not investigate further into this here, but just note that many writers perceive that the changes are happening faster today due to the rapid technological development [135, 256] and the large changes within the economy and society [256]. This also influence the social construction process, since there are more actors directly or indirectly influencing the organizational reality. An introduction to the theory on the logic of organizational change can be found in Morgan [227] and Narayanan and Nath [233]. The issue of changes in computerized information systems in particular is discussed by Lehman [189] and Perry [255].

As illustrated above, a computerized information system is embedded in a cultural matrix of applications, users, laws, and hardware which all change continually. Thus, all successful software get changed, there is nothing detrimental about this. Change is the norm, not the exception for large information systems [2, 334]. A first step towards facing this is to accept change as a way of life, rather than as an untowarded and annoying exception.

1.3 Chapter Summary

We have in this chapter given a brief introduction to the philosophical outlook for our work:
- Organizations are realities socially constructed through the joint actions of its social actors.
- Change is the norm, not the exception for organizations and their application systems.

We will return throughout the thesis to how this outlook influences conceptual modeling for computerized information systems support in organizations.

Chapter 2

Computerized Information Systems

Most organizations of some size will have a partly computerized information system (CIS). The main official motivation for having this is to support adequate action so that value addition may be achieved. The degree of computerization and how CIS-support is performed in a specific organization on the other hand, differs widely from organization to organization.

We will in this thesis concentrate on the development, use, and maintenance of application systems in technical and administrative organization. Such CISs usually differ from real-time embedded control systems in that they are more strongly characterized by relatively simple functional demands, massive size, large volumes of data, continuous use of a multitude of users, and continuous evolution [2], although you might also find real-time systems that have one or more of these characteristics.

We have chosen to present the area by first outlining the terminology which will be followed throughout the thesis. The emphasis on terminology is motivated by the current situation of a set of fuzzy and ill-defined terms in the area [174, 195]. Thus, we find it convenient to define some essential terms which will be used frequently in the rest of the thesis explicitly.

After presenting the terminology, we will describe some perceived problems with CISs in organizations.

2.1 Terminology

We will in this section present main parts of the terminology which we use to discuss CIS-support in organizations. Additional definitions will be presented in the thesis as appropriate. A comprehensive overview of our terminology is given in Appendix H. An index of the terms used in the terminology is given in the end of this appendix.

Terms will in this section be written in **bold** type when first defined, and will be written *slanted* when they are used as part of other definitions. The terminology is in particular based on [195, 293, 323]. It is also influenced by our philosophical stance of social construction discussed in Chapter 1.1. This appears especially in the definitions of the basic terms, although the definitions themselves are written in a categorical style for us to be able to use the terms consistently. A constructivistic view is also followed in FRISCO, a group within IFIP WG 8.1. on design and evaluation of information systems which are trying to establish a framework for information systems terminology [95, 195]. Similarly to FRISCO, we have used a set-theoretic approach.

The terms are grouped according to the following areas:

- Phenomena and states.

- Data, information, and knowledge.
- Language and models.
- Actors and activities.
- Systems.
- Methodology.

2.1.1 Assumptions underlying the Terminology

- The world is perceived by any person to exist in time and space.
- There is a set of persons in the world, each which perceive the world individually, and thus potentially differently.
- For **society** it is of prime interest that one creates inter-subjective agreement between persons on their perception of parts of the world.
- A process of social construction takes place within organizations as described in Chapter 1.1, and we will use the terms that were introduced there without further explanation.

2.1.2 Phenomena and States

A phenomenon is used as the elementary unit of the terminology. In other terminologies, the term 'thing' is used in this respect [195, 323].

Phenomenon: A phenomenon is something as it appears in the mind of a person. The world is perceived by persons to consist of phenomena. A phenomenon can be perceived to exist independently of the perceiving person, or be perceived to be a purely mental construction.

A *phenomenon* is of **relevance** to a non-empty set of persons in a *time interval* if it is of interest to all members of the set in the *time interval*. A *phenomenon* is of **potential relevance** to a non-empty set of persons in a *time interval* if it is of interest to at least one of the persons in the set in the *time interval*.

Relevance is socially and temporally constrained which are as expected taking social construction into account. Relevance needs the notion of *shared explicit knowledge* to be meaningful i.e. if there are no phenomenon which are perceived equally by two persons, no phenomena will be relevant.

Property : A property is an aspect of a *phenomenon* which can be described and given a value. A phenomenon will have a set of *potentially relevant properties*. The values for the properties are members of the *domains* for these properties. A **domain** is defined as the source of any kind of mapping.

'Domain' includes the meaning known from algebra, but the term 'mapping' is used in a slightly more general sense than usual. Not only sets can be mapped into sets as in mathematics, but also areas into areas. When used in the mathematical sense, a domain will be a finite or infinite set of values.

All *phenomena* have at least one property, namely its perceived existence or lack thereof.

State, transition, and event: The state of a *phenomenon* is the set of mappings of all *properties* of the *phenomenon* into values from the *domain* of the *properties*. A *phenomenon* can only be in one state within a *time unit*. A **transition** is a mapping from a domain comprising

states to a co-domain comprising *states*. An **event** is a change of *state* of a *phenomena*. It is effected through a *transition*. An event happen within a *time unit*, i.e. it has a zero *duration*. The **history** of a *phenomenon* is the chronologically ordered *states* of the *phenomenon*.

2.1.3 Data, Information, and Knowledge

Knowledge : Knowledge is that which is accepted as valid, relevant and true by a person. Knowledge is by definition linked to the individual. It can be divided into *explicit* and *tacit knowledge*. **Explicit knowledge** is the awareness of a person of *properties* and values of *properties* of *phenomena*.

The explicit knowledge of a person can be looked upon as part of the *state* of this *phenomenon* since a *potential relevant property* of a person can be that he is able to know something, and what he or she is perceived to know.

Shared explicit knowledge is an inter-subjectively agreed identical awareness of some *properties* and values of *properties* of *phenomena* by two or more persons which have been achieved through a process of social construction.

Tacit knowledge is *knowledge* that can not be represented externally to the person and only shows up in *action*.

Information : Information is externalized *explicit knowledge* which is not already known by the person who receives it, i.e. a *state transition* for a person appears when he receives information, thus receiving information can be looked upon as an *event*.

This means that 'information' is socially and temporally constrained. If I already know something (and know that I know it) , I do not perceive to receive information if I am told the same thing again (even if the certainty of the knowledge might increase). Thus our definition is hopefully close to the one used in everyday language as illustrated in [195] "Information is what you get or may get if you ask certain kinds of questions.... Answers to such questions are often provided at some information desk."

Symbols and data : The *explicit knowledge* of a person can be externalized in a persistent form using symbols. Data are *symbols* that can be preserved, transformed, and transported by a computer. Data and other *symbols* can be internalized as *knowledge* by persons.

2.1.4 Language and Models

Language : A set of *symbols*, the **graphemes** of the language being the smallest units in the writing system capable of causing a contrast in meaning, a set of **words** being a set of related *symbols* constituting the **vocabulary** of the language, rules to form **sentences** being a set of related words (**syntax**), and some inter-subjectively agreed definitions of what the different sentences mean (**semantics**).

In a natural language, e.g. English, the symbols and words will be ordered linearly, whereas in a diagrammatic language symbols are ordered spatially. In addition to the aspects described above, one also often talks about the **pragmatics** when discussing languages, being the relationship between symbols, words, and sentences and the effect these have on persons.

A **formalism** is a formal *language*, i.e. a *language* with a precisely defined *vocabulary*, *syntax*, and *semantics*. A **semi-formal language** is a *language* with a precisely defined *vocabulary* and *syntax*, but without a precisely defined *semantics*.

Abstraction : An abstraction is the phenomenon of a set of *phenomena* and its *properties* at some level of approximation. The abstraction contains incomplete *explicit knowledge* about the *phenomena*.

Model : A model is an *abstraction* externalized in a *language*. A model is assumed to be simpler than, resemble, have the same structure and way of functioning as the phenomenon it represent. A **conceptual model** is a *model* of a *domain* made in a *formal* or *semi-formal language*. Many conceptual modeling languages are partly diagrammatic, in which case they are logographic, but this is not looked upon as a requirement. On the other hand, we will primarily look upon models made in a diagrammatic languages, having a limited set of words and syntax-rules for creating sentences being meant to be applicable for modeling in a large range of areas in this thesis.

2.1.5 Actors and Activities

A *phenomenon* is **acted** upon by another *phenomenon* if its *history* is different from what it would have been if the other *phenomenon* did not influence it.

Actor : An actor is a *phenomenon* that *acts upon* another *phenomenon*, the **actand**.

Main classes of actors are illustrated in Figure 2.1.

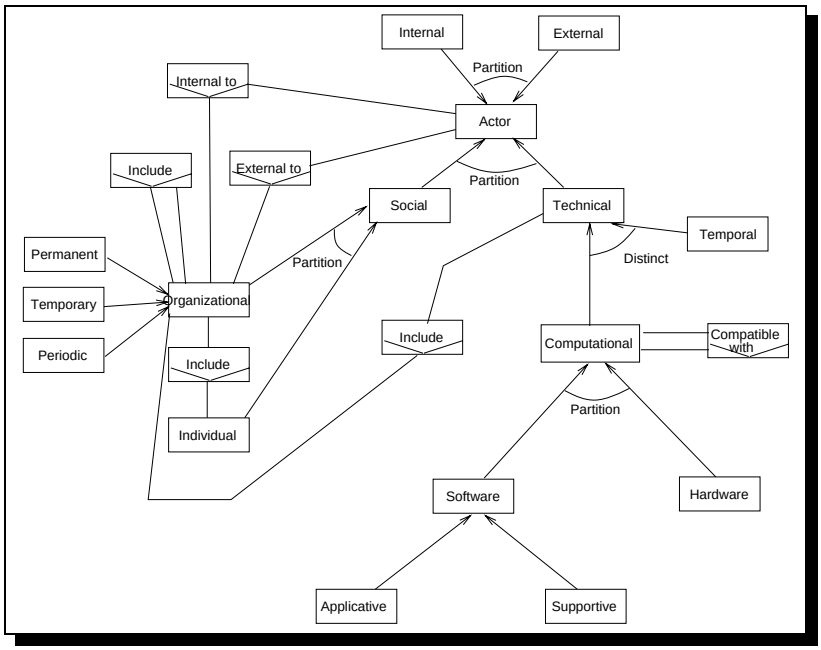

Figure 2.1: Actor class hierarchy

A **social actor** is an *actor* that includes at least one person. Social actors might be **individual** or **organizational**. A **technical actor** is an *actor* that do not include any persons.

Technical actors can be **computational** or **temporal**. Whereas temporal actors are some time-measuring device (i.e. a clock of some sort), computational actors are either **hardware** or **software** actors. Computational actors can be said to be compatible in the following meanings:

- **Hardware compatibility**: Stating which *hardware actors* that can *act* upon each other.
- **Executional compatibility**: Describe which *software actors* that can be executed on which *hardware actors*.
- **Software compatibility**: Stating which *software actors* that can *act* upon each other.

Software actors are **supportive** or **applicative** relative to an *organization*. The difference is that applicative actors are being customized to some degree to cater for the specific needs of the organization.

An **internal actor** relative to an *organization* is an *actor* being part of the *organizational system* of the *organization* in one or more of the relevant *roles* it is currently filling. An **external actor** relative to an *organization* is an *actor* not being part of the *organizational system* of the *organization* in any of the relevant *roles* it is currently filling.

A person interacting with his *environment* is termed an **individual social actor**. An **organizational actor** is a *social actor* which consist of a set of more than one person performing goal-oriented and co-ordinated action. An organizational actor can also include *technical actors*. We distinguish three types of organizational actors:

- Permanent: Organizations that are meant to exist for an indefinite time.
- Temporary: Organizations that are meant to have a limited life-span.
- Periodic: Organizations which have a limited life-span, but which reappear at periodic intervals. The **reincarnation** of periodic organizations usually depend on the existence of a permanent organization.

Action : An action is the *phenomenon* of one *phenomenon acting* upon other *phenomena*. An **activity** is a *system* of *actions*.

Role : *Behavior* that can be expected of an *actor* by other *actors*. An *actor* acting in a particular *role* is termed an **agent**. A **formal role** is a *role* where part of the expected *behavior* of an *actor* filling the *role* is institutionalized by an *organizational actor*. All roles have usually also two additional aspects:

- The informal part of the *role*. Expectations to an *actor* filling the *role* which are not institutionalized.
- The expectation to an *agent*, because of the particular *actor* filling the *role*.

2.1.6 Systems

System : A system is a set of correlated *phenomena*, which itself is a *phenomenon*. Each *phenomenon* that is contained in the system is said to be **part** of the system. A system has at least one systemic *property* not possessed by any of its parts. A **subsystem** of a *system* is a *system* that is part of another *system*, the set of *phenomena* being part of the subsystem is a proper subset of the set of *phenomena* being part of the whole *system*.

Information system (IS) : An information system is a *system* for the dissemination of *symbols* between persons, i.e. to potentially increase their *knowledge*.

A **data system** is a *system* to preserve, transform, and transport *data.* A *data system* is usually a sub-system of an *information system.* Both *data systems* and *information systems* may be contained in the *domain* they convey *data* about.

Organization : An organization is defined as a non-empty set of persons, and other *phenomena* which are a *phenomenon* where goal-oriented and co-ordinated *action* is aimed at. An organization is an *organizational actor* when interacting with other *phenomena.* An **organizational system** is a *system* having the *actors* and *activities* of an *organization* as *subsystems.*

Organizational information system (OIS): An OIS is the *information system* for the dissemination of *data* within an *organization.* The OIS is a *subsystem* of an *organizational system.*

Computerized information system(CIS): A CIS is an *information system* which are based on the use of computers for the dissemination of *data.* A **computerized organizational information system (COIS)** is a *system* for the dissemination of *data* within an *organization* which are based on the use of computers. This is a *subsystem* of the *OIS* of the organization.

The COIS is the set of *internal software actors* which support the *internal social actors* of the *organization,* and the *hardware actors* these *software actors* are executed on.

Application system : An application system is a *subsystem* of the *COIS* being adapted to the needs of the *organization.* When the application system interact with its *environment* it is an *applicative actor.* The **(application system) portfolio** of an *organization* is the set of *application systems* in the *COIS* of the *organization.*

2.1.7 Methodology

The terms are defined here in the context of conceptual modeling for CIS-support in organizations.

Development (of an application system in an organization): The *process* of producing a new *application system* in the *organization* based on the current *OIS* and the *knowledge* of *internal* and potentially *external actors.*

Development is divided into two categories.

- Development of **replacement systems** being *application systems* that replaces existing *application systems,* and offers the same functionality as the already existing *application systems.*
- Development of *application systems* covering functional areas that is not currently supported by the existing *COIS.*

Maintenance (of an application system in an organization) : The *process* of creating an updated *version* of an *application system* in the *organization* through a temporally ordered set of *lawful transitions* based on an existing *application system* and the *knowledge* of *internal* and potentially *external actors.*

Maintenance of application systems has traditionally been divided into three types, corrective, adaptive, and perfective [298].

- **Corrective maintenance:** Performed to identify and correct processing failures, performance failures and implementation failures in an existing *application system.*

- **Adaptive maintenance:** Performed to adapt *application systems* to the changes among the *supporting technical actors* of the *application system.*
- **Perfective maintenance:** Performed to enhance performance, change or add new functionality, or improve future maintainability of the *application system.* Perfective maintenance can be divided into functional and non-functional perfective maintenance based on the effects of the performed changes. Functional changes implies changes to the functions offered by the *application system* or said differently, how the *end-users* can potentially increase their *knowledge* using the *application system.* Non-functional changes implies changes where the quality features of the *application system* and other features being important for the developer of the *application system* such as modifiability is improved.

Some also use the term preventive maintenance, i.e. the work of making application systems more maintainable through for instance restructuring. This kind of maintenance has been included as part of non-functionally perfective maintenance above. The above typology of tasks covers the different changes to a COIS being discussed in Chapter 1.2.

In addition to the traditionally temporal distinction between development and maintenance indicated above, we have introduced the terms 'functional maintenance' and 'functional development' [173, 181]:

- **Functional development:** *Development* or *maintenance* where changes in the *application* increases the functional coverage of the *portfolio* of the *organization.* This includes *development* of new *application systems* which covers areas which are not covered by the existing *COIS*, and also includes *functional perfective maintenance.*
- **Functional maintenance:** Work made to keep up the functional coverage of the *portfolio* of the *organization.* This includes the three other types of *maintenance*, but also includes the *development of replacement systems.*

This distinction is introduced because it is believed to give a better indication of the efficiency of the computerized information systems support in an organization [173] than the usual distinction between development and maintenance taking the ever-changing nature of organizations as described in Chapter 1.2 into account. The relationship between the areas of traditional development and maintenance and functional development and maintenance is shown in Figure 2.2.

Devtenance in an organization : The *process* of producing an updated *version* of the *COIS* through a temporally ordered set of *lawful transitions* based on the existing *OIS* and the *knowledge* of *internal* and potentially *external actors.*

Devtenance includes both traditional development and maintenance activities and was originally defined as [169]:

"A more comprehensive life cycle that includes maintenance, uniting software development and maintenance functions by sharing the same tools and methods in both."

Methodology: A *system* of *rules, techniques*, and tools to aid *development and/or maintenance* of *application systems.*

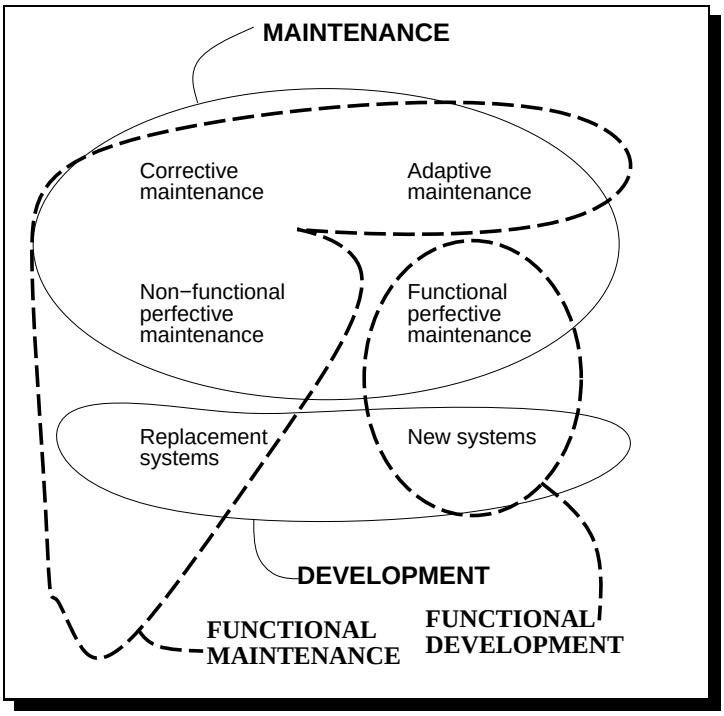

Figure 2.2: Relationship between functional and traditional development and maintenance

2.2 Problems with Computerized Information Systems

Since the application of computers in administrative data processing began in the fifties, computers have become a key instrument for automated data processing in organizations. However, even if this rapid development has given us a powerful tool for organizational problem solving, many empirical investigations have found CIS-support to be rife with difficulties [165, 206, 311].

This has led many researchers in the past to speak of a "software crisis", a term which is still in use [101]. The cost and time overrun of development projects [100, 149, 260] and the large amount of resources used on maintenance [113] have been claimed to be typical symptoms of this crises. That difficult tasks often fail to meet their cost and time schedule is quite usual in many areas of complex human endeavour. As will be discussed in Chapter 3, looking on the amount of resources used on maintenance is neither a good indicator, due to the large number of "new" application systems being replacements [173, 299]. On the other hand, a deep dissatisfaction with the CIS support seems to exist in many organizations.

2.2.1 Why is there a Problem?

Several researchers have tried to explain the "software crisis" emphasizing that development and maintenance of a CIS can be considered especially difficult. Two frequently cited treatments of this are those given by Rittel [268] and Brooks [35], and we will briefly discuss their views in the light of our philosophical outlook.

Problems with the support of CISs in an organization are often *wicked problems* [268]. The main properties of wicked problems are:

- Wicked problems have no definitive formulation. You cannot understand the problem without solving it, and solving the problem is the same as understanding it.

The different stakeholders in the problem have their individual local reality which are used when assessing what the problem is about, i.e. they might look upon the problem differently. Under the process of solving the problem, both the external environment of the organization, and the way the individuals look upon what the problem is might change during the sense-making process, thus a definite formulation of the problem is not possible to achieve.

- Every formulation of a wicked problem corresponds to a statement of its solution.
  When making a problem statement in the first place, this is trying to externalize the local reality of one or more persons, and as such might influence the way other persons in the organization look upon the problem by closing the space of possibilities.
- Wicked problems can always be solved better, there is no stopping rule for wicked problems.
  First, even if a solution is regarded to be good by one of the stakeholders, it is not necessarily looked upon as good by another. Secondly, as part of the problem solving process, the local understanding of the problem will change as part of the sense-making process, opening for a wish to perform additional externalizations.
- Solutions to wicked problems are neither correct nor false, we can only say that they are good or bad and this to a varying degree and maybe in different ways for different people.
  Also this can be based on the local reality which the evaluation of the externalized solutions is based on. Since there do not exist one true solution to the problem, this can obviously be explained by the constructivistic stance.
- In wicked problems there are many explanations for the same discrepancy and there is no test which of these explanations are the best one.
  Since people make sense of the world based upon their local reality which are bound to differ, there is not necessarily any single explanation on a given problem, and by the outset no implicit understanding that a given explanation is the best.
- Every wicked problem can be considered a symptom of another problem.
  The present situation is based on the previous externalizations by different organizational actors, and the present understanding of the problem by the individuals are based upon the internalization of previous externalizations. What is looked upon as a problem can thus be caused by the application of solutions to earlier problems, that addressed a situation different from the one that is currently present or which is no longer perceived as important.
- Every wicked problem is essentially unique and solving it must be regarded as a one-shot operation. There is no exhaustive, enumerable list of permissible operations, everything goes as a matter of principle and fantasy.
  The uniqueness stems from the uniqueness of the internal reality of the present stakeholders of the problem. The situation in one organization will always be unique compared to what has appeared other places, and also compared with an earlier situation in the same organization.

Brooks [35] looks on the area from a more technical point of view and claim that the problems stems from what he calls the *essential difficulties* of CISs. These are:

- Complexity: CISs are more complex for their size than other man-made systems. Application systems usually have a much larger state space than other kinds of man-made systems and there is no repetition of components.
- Invisibility: Software artifacts are invisible and impossible to visualize. The reality of CISs is not inherently embedded in space. This not only impedes the process of design within one mind, it severely hinders communication among minds.

The above two points can not be said to be directly related to social construction and continuous change. On the other hand, it points to an important problem with CISs in an organization. The externalization in the form of a CIS is very difficult to make sense of for the organizational actors, and thus very difficult to change in a direction that is perceived to be useful as the environment change and internalization based on other externalizations take place. The reason that it is possible to create system with such a high complexity as modern CISs have, is the abstraction mechanisms that are used. But these are often not perfect, side-effects of what is done are often present. In addition, abstraction implies a loss of meaning, when abstracting on a too high level, no-one, neither the developers nor the users grasp all the consequences of the model.

- Conformity: There are few firm principles for the development and maintenance of CISs. Much of the complexity is arbitrary complexity, forced without rhyme or reason by the many human organizations and other systems it must conform to.
  Since the development and maintenance of a CIS is the externalization of the local reality of the organizational actors, this follows directly from social construction theory.
- Changeability: CISs are more often required to change than other kinds of systems. This because they encode parts of the world that often change, like business policies and working routines. In addition, they are by many users perceived as easy to change, it is pure thoughstuff, infinitely malleable.
  This is directly related to the aspect of continuous change. Combined with the complexity, malleability, and invisibility of software, this creates additional problems. In contrast with normal industrial practice where conceptual changes are inputs to a redesign and recreation process that ultimately produces a new instance of the system, changes to CISs are superimposed on a current embodiment [189].

Whereas some of the problems with CISs are due to the properties of CISs themselves such as complexity and invisibility, many of them seem to be due to the inherit complexity of the organizational reality that they are supposed to function within. Many of them are not technical, but rather social and without methodologies that take this into account one can not hope for substantial improvements in the perceived quality of CIS support. Note that it is not only the devtenance of CISs that has to deal with a socially constructed ever-changing world. It is combined with the unique technical possibilities and problems of CISs this turns out to be an especially demanding area.

2.3 Chapter Summary

We have in this chapter presented the basic terminology being used in the thesis. This is meant to be helpful for the prospective reader of the thesis in comprehending the ideas externalized on these pages. A more extensive terminology is presented in Appendix H.

Overall problems that appear to hamper CIS support in organizations as discussed by Rittel and Brooks were investigated in the light of our philosophical outlook from Chapter 1, and it was concluded that it seems to be the combined effect of aspects related to the philosophical outlook and the unique technical opportunities and problems of CISs that make COIS-support especially hard.

In the next chapter a closer overview over how development and maintenance of the COIS is performed in organizations is given.

Chapter 3

Development and Maintenance of the COIS

We will in this chapter introduce the state of practice of CIS-support in organizations as we perceive it based on different case-studies and survey-investigations. The chapter will concentrate on the presentation of a survey which we have performed among Norwegian organizations, but will also refer to work in this area performed by other researchers.

Obviously, results from such investigations do not in any sense reveal the "true" state of business when it comes to CIS support in organizations. What they *do* give, is an indication of the general problems that organizations seem to face on this matter and the techniques that are presently being used to answer these problems. Especially when similar patterns are found in several investigations, these indications can be useful for our understanding. This chapter together with our philosophical outlook is thus meant to set the stage for our more theoretical results.

3.1 Introduction to the Survey Investigation

Even if many surveys have been conducted in the area of CIS support in organizations, both on development (e.g. [24, 100, 149, 315]) and maintenance (e.g. [69, 159, 193, 237, 299]), many areas have not been thoroughly investigated in Norwegian organizations. Our survey investigation took place in 1993, and the figures reported are based on the responses from 52 Norwegian organizations. Even if both development and maintenance practice were surveyed, the emphasis in the survey was on maintenance. A more detailed overview of how the investigation was conducted is given in Appendix F.

Only results deemed relevant to the rest of the thesis will be given here. A more complete report from the survey has been issued [171]. Selected results have earlier been presented in [170, 173, 175, 176, 181]. We will in the following both present general descriptive results, in addition to the results of statistical analysis of the data. Significant results are generally shown using **bold** type. A description of the analysis being done and the rationale for choosing these can be found in Appendix F.4. Whereas we in the papers have presented the results based on the rejection or acceptance of stated hypothesis, we have chosen to only present the main results here to limit the size of this chapter.

The investigation had two main motivations:

- To compare the development and maintenance situation in Norway with what has been reported in similar investigations from other countries, and thus to what extent we can assume

that the problems reported in other investigations also are relevant in Norway, and if the novel results found in our investigation can be expected to have more general applicability.
- To investigate new areas beyond what is reported in literature in order to better be able to assess the efficiency of the CIS-support in an organization.

The numbers reported in this chapter are from our survey if nothing else is indicated. In addition the following investigations are referred to extensively below.

- The Lientz and Swanson investigation [193]: The investigation performed by Lientz and Swanson in 1977 is the most comprehensive study of software maintenance in data processing organizations to date, with responses from 487 American organizations.
- The Nosek and Palvia investigation [237]: A followed-up study to Lientz/Swanson performed in 1990 asking many of the same questions. Their results are based on responses from 52 American organizations.
- The Swanson and Beath investigation [299]: A case-study among 12 American organizations in the late eighties which in addition to questions given in the Lientz/Swanson study focused on portfolio analysis and replacement systems.
- The Bergersen investigation [24]: A combined case-study and survey investigation on application systems development in Norwegian organization being performed in 1990. Bergersen based his statistical analysis on the reports from 106 organizations.

3.2 Computerized Information System Support in Organizations

The organizations responding to our survey were in (no of cases in parenthesis): Manufacturing and industry (20), public services (7), insurance and banking (8), trade (6), others (11). 80 of the organizations had a yearly CIS-budget above 5 million Nkr. The descriptive statistics of the size-measures used for the respondents are given in Table 3.1.

Figure	N	Range	Mean	Median	SD
Number of employees	52	[20-35000]	2347	555	6499.54
Employees in data department	52	[1-250]	24.3	10	40.64
Number of systems developers	52	[0-87]	9.5	5	16.33
Number of major systems	51	[2-100]	10.3	5	18.86
User population	50	[20-5000]	541	250	883.80

Table 3.1: Descriptive data for size-measures

None of these size-measures were found to be normally distributed, so for statistical analysis, the logarithm of these numbers was used. A conversion to a logarithmic form is often used. Such conversions are often used since it normalises the distribution. Tests for normality for the normalized figures are given in Table 3.2.

The results do not give us any reason to reject the null-hypothesis that the numbers for the size measures, except for on the number of major application systems, are normally distributed. For the number of major applications systems, only non-parametric tests are used. None of the

Figure	Skewness	Kurtosis	Shapiro-Wilks Sign.		(K-S) Lilliefors Sign.	
Log(employees)	.5250	.3469			.0897	>.200
Log(data department)	.1492	.1174			.0763	>.200
Log(systems developers)	.5724	.5284	.9507	.0796	.0954	>.200
Log(major systems)	.3896	.5085	.9489	.0618	.1332	.0296
Log(user population)	.1979	-.1786	.9774	.6236	.0624	>.200

Table 3.2: Tests for normality of size measures

distributions are perfectly normal though, since the kurtosis and skewness are different from zero, but this would be expected even for a sample from a normal distribution [236].

3.2.1 Distribution of Effort on Development and Maintenance

The amount of maintenance compared to the amount of development effort has been measured in a number of investigations, and in most investigation somewhere between 40-60% of the effort is found to be used on maintenance on average [113, 181].

Table 3.3 summarizes the descriptive results on the distribution of work in our survey, splitting it up on mean value with accompanying standard deviation for the different categories, and giving the mean value for the figures according to the assessed quality of the answers.

The first set of figures in Table 3.3 list the numbers as reported. The second list reports them disregarding other work than development and maintenance of information systems. The next figures gives the work on traditional maintenance split up in separate categories for the whole portfolio whereas the next shows the same numbers reported on individual systems. In the following discussion, the numbers given for the whole portfolio are used. After this follows figures for the development activity. The last set of figures in the table shows the amount of functional development and maintenance based on the data for the whole portfolio.

Comparisons of descriptive results are presented in Table 3.4. The amount of *other work* reported in our investigation is much larger than in the other investigations. Therefore comparisons are done without taking this into account. Before using hypothesis testing, the figures were tested for normality as illustrated in Table 3.5. The results do not give us any reason to reject the null-hypothesis that the numbers for both traditional maintenance and functional maintenance are normally distributed.

Table 3.6 sums up the comparison with the American investigations. When disregarding other work, we see that our result of 59% is not statistically different from what was reported in Lientz/Swanson and Nosek/Palvia. This is contrary to the popular belief that the maintenance proportion of software expenditure is rising, as also pointed out by Foster [93]. This gives us increased confidence that the values we have found for the amount of functional maintenance and development give a correct impression.

From Table 3.7 we see that the amount of functional maintenance is significantly smaller than the amount of traditional maintenance and vice versa when it comes to development. The amount of functional maintenance is significantly smaller than the amount of functional development, but is still as high as 44%. Almost half the work that is done on CIS support is done without enhancing the functionality of the systems.

Category	Mean	SD	Good	Estimate	On data	Guess
Figures as percentage of all work performed by data processing department						
Work on development	30	20.31	41	23	29	30
Work on maintenance	40	23.73	35	46	42	38
Other work	30	29.46				
Figures as percentage of the work done on development and maintenance only						
Work on development	41	24.17	53	34	41	42
Work on maintenance	59	24.17	48	62	57	61
Maintenance by category for the whole portfolio, average quality = 2.2						
Corrective maintenance	26	20.96	27	28	28	23
Adaptive maintenance	10	12.30	12	7	9	11
Functional perfective maint.	51	23.86	54	57	56	45
Non-functional perfective maint.	13	15.57	7	7	7	21
Maintenance by category for one system, average quality = 2.3						
Corrective maintenance	15	16.33	8	19	18	11
Adaptive maintenance	5	8.92	1	6	6	5
Functional perfective maint.	60	26.03	79	57	59	60
Non-functional perfective maint.	20	17.54	12	18	17	24
Development by category for the whole portfolio, average quality = 2.1						
Replacement	38	28.15	31	47.5	42	30
New development	62	28.15	69	52.5	58	67
Figures for functional development and functional maintenance						
Functional development	56	17.89	62	53	56	55
Functional maintenance	44	17.89	39	43	42	46

Good (1) The number is based on good data.
Estimate (2) The number is based on minimal data.
Guess (3) The numbers is a guess not based on data.
On data Includes the **Good** and **Estimate** columns.

Table 3.3: Distribution of work effort

Category	Own		Lientz/ Swanson		Nosek/ Palvia	
	Mean	SD	Mean	SD	Mean	SD
Percentage of all work						
Development	30	20.31	43	22.3	58	25
Maintenance	40	23.73	49	21.1	35	23
Other work	30	29.46	8	13.0	7	13
Disregarding other work						
Development	41	24.17	47	>22.3	38	>25
Maintenance	59	24.17	53	>21.1	62	>23
Maintenance by category						
Corrective	26	20.96	20			
Adaptive	10	12.30	25			
Functional perfective	51	23.86	42			
Non-functional perfective	13	15.57	13			

Table 3.4: Comparing the work distribution in various investigations

Figure	Skewness	Kurtosis	Shapiro-Wilks	(K-S) Sign.	Lilliefors Sign.
Maintenance	-.1111	-.8133	.9559	.1386	.0706	>.200
Functional maintenance	.1513	-.0974	.9728	.4770	.0701	>.200

Table 3.5: Tests for normality of maintenance figures

As shown in [175] both maintenance measures are negatively correlated with most of the size measures, which again are highly intercorrelated. We will not discuss the connection between the size and effort measures in detail in the thesis.

The distribution of time used in different categories of maintenance work in our investigation is also comparable to what has been reported by others, except that the organizations in our survey seem to use less time on adaptive maintenance, but more on corrective and functionally perfective maintenance than those of the Lientz /Swanson investigation. The distribution of time used on different categories of maintenance work is almost identical to the numbers presented by Henne [130].

Only 6% of the effort on single systems was used to perform emergency fixes, the similar figure in Lientz/Swanson being 12%. The total amount on corrective maintenance on the individual systems was 15%. Jørgensen [158] indicate that the assessed corrective percentage of the work used on maintenance often might be exaggerated since these kind of problems are more visible for management. They found in their investigation of individual maintenance tasks that even if 38% of the changes were corrective, this took only up 9% of the time used for maintenance. Management assessed the percentage of corrective maintenance to be 19%, Those managers who based their answers on good data had a result of 9% corrective maintenance. Also in our investigation,

Figure	Δ mean Lientz/ Swanson	Statistical significance	Δ mean Nosek/ Palvia	Statistical significance
Maintenance	-9	**p<.02**	-18	**p<.01**
Development	-13	**p<.01**	-5	No
Other	22	**p<.01**	23	**p<.01**
Maintenance only	6	No	-3	No
Development only	-6	No	3	No

Table 3.6: Testing of equality of work distribution

Figure	Δ mean	Statistical significance
Functional vs traditional maintenance	15	**p< .01**
Functional vs.traditional development	15	**p< .01**
Functional maintenance vs.functional development	12	**p< .03**

Table 3.7: Testing relationships between functional and traditional maintenance measures

we found a similar tendency, on the data of the maintenance task of the individual systems, those reporting to have good data, reported that only 8% of the work effort was corrective maintenance, 4% being emergency fixes. The same effect on over-assessing the amount of corrective maintenance has been reported earlier by Arnold [10]. The effect of this would then probably decrease the total amount of functional maintenance even further, thus indicating an even better situation regarding average functional maintenance.

3.2.2 Organization of Development and Maintenance

The development of an application system is usually organized as a project, meaning that it is supposed to take a limited time before the resulting system is put into use, it supposedly has an unambiguous goal and is a unique, essentially un-repeatable task . Maintenance on the other hand is often performed on a more problem to problem basis, although 40% of the respondents in the investigation reported that except from emergency fixes, all changes to the application systems are batched for periodic implementation in a project-like manner.

Several investigation indicates that a minority of new application systems are delivered within scheduled time and budget. Bergersen found in his investigation that only a third of the systems had been delivered within time and on or below budget. Jenkins et al. [149] reported an average effort overrun of 36% and average schedule over-run of 22% in their investigation.

In Bergersen, the three most important factors for overall perceived project success were found to be the goal-setting, management support, and user-participation. In van Swede [315] the main contributions of success in the sense of satisfaction of all stakeholders were a cooperative environment, presence of a win-win starting point by considering the interest of all stakeholder-group,

quality of project staff, and quality of project management.

The system developers and maintainers in an organization can be organized in several ways. One of the main results of Lientz/Swanson was that the amount of time used on maintenance was statistically significantly less in the organizations where development and maintenance were organized separately. 16.2 % of the organizations reported to have this kind of departmentalizations in Lientz/Swanson whereas only 8% of the organizations in our investigation used this kind of departmentalization. The opposite result was found in our investigation, the organizations with this kind of departmentalization used *more* time on maintenance than those not having this split. Because of the small number of organizations that had a separate maintenance division though, we should be cautious in the interpretation of this especially because of the big difference in variance between the two groups. When using a non-parametric test, no significant results were obtained. Neither could we observe any significant difference in amount of the functional maintenance.

As part of development and maintenance the developers are producing different artifacts. This is sometimes done individually, but most often includes a certain amount of collaboration and coordination between the team members which implies a need for good communication structures. The team can be geographically distributed or co-located. The team will typically evolve, meaning that the team will consist of different persons at different times. The team is typically disbanded after that the development is regarded as finished. On the other hand it is usual that some of the same persons that developed the application system continuous with the maintenance of it. On the question "Is maintenance of the application system performed by those who made the original system?" the below distribution of answers was returned from the organizations not having an organizational separation of development and maintenance:

Always |**8**| **5** |**21**| **4** |**14**| **3** |**2**| **2** |**1**| **1** **Never**

Number of responses: 46, Mean 3.7, Median 4

Thus even if persons change jobs, and external actors are used for maintenance in some organizations, some of the same persons who developed the application system originally also perform the maintenance of it in most cases.

3.2.3 Methods and Tools

31% of the organizations claimed to use a complete development methodology covering all phases of development and maintenance. All organizations having a complete development methodology had a CIS-budget of more than 5 million Nkr. The descriptive statistics for the number of employees in these organizations were:

Range [70-35000], Mean 5858, Median 1150.

Comparing to the average in Table 3.1, we see that it is primarily organizations above average size that have a complete development methodology installed. Dividing the sample according to if the organization had a complete development method or not and investigating on the connection with the maintenance figures and the size indicators is given in Table 3.8. Whereas most of the size measures were significantly higher and also the use of organizational controls was, only functional maintenance of the maintenance measures was significantly smaller in the organizations claiming to have a complete methodology.

The term organizational controls refer to certain routines being institutionalized as part of maintenance. Table 3.9 indicates the use of such controls.

Figure	Using methodology			Not using methodology				
	N	Mean	SD	N	Mean	SD	Δ	p
Maintenance	14	50	25.929	33	60.9	22.215	-10.9	.150
Functional maintenance	14	34.2	13.947	31	47.1	17.253	-12.9	**.018**
Log(employees)	15	3.0834	.792	34	2.4813	.556		**.004**
Log(data department)	15	1.4526	.464	34	.8697	.451		**.000**
Log(system developers)	15	.9901	.471	30	.6204	.357		**.005**
Log(major systems)	14	.8266	.228	33	.7118	.254		.151
Log(user population)	15	2.6499	.548	32	2.2651	.463		**.016**
Organizational controls	14	7.8	2.563	33	5.5	2.563	2.3	**.005**

Table 3.8: Maintenance and size vs. use of overall methodology

Compared to Lientz/Swanson and Nosek/Palvia, the pattern of use of organizational controls appears to be somewhat different in the Norwegian organizations [181]. Some areas, like cost justification, re-testing of changes and batching of changes appears to be better taken care of in Norway than in America. On the other hand, such things as logging of user request, logging of changes and performing periodic formal audits seem to be better taken care of in the American organizations. A comparison between which controls were institutionalized, and the use of similar controls in the maintenance of a single application system gave that they did not seem to have a significant impact on the behavior of keeping the users informed about the status of change requests and on updating the documentation of the programs when they are changed. Apart from this the institutionalizing of controls influenced the way maintenance and handling of change-requests was done.

CASE-technology was used by 27% of the organizations for development and 10% of the organizations for maintenance. All organizations using CASE had more than 5 million Nkr in the IS-budget. The descriptive statistics for the number of employees of the CASE-users were:

Range [150-35000], Mean 6645, Median 1000.

Comparing to the average in Table 3.1, we see that it is organizations of above average size that applies CASE-technology. 77% of the organization using CASE-technology had a complete development methodology. The CASE-users not having a complete development methodology were the smallest organizations in terms of number of employees that used CASE. We would thus expect somewhat the same pattern as on development methods above. Table 3.10 gives an overview. We see that it was basically large organizations who had adopted CASE-tools. Even if the amount on maintenance and functional maintenance was less when using CASE-tools, even if not significant, we do not believe that it is the use of CASE that is the reason for this, but rather that CASE is being taken into use in large organizations with an already existing overall methodology and a sound view on system development and maintenance. This is based on the answers reporting on the impact of CASE which indicated an average use of CASE of 2.8 years and only 10% of the application systems in the portfolio being supported by CASE. Looking on the individual organizations the number of systems supported by CASE varied between 0 and 50% of the major

Control	N	%
All changes to the application programs undergo a formal retest procedure	38	79.2%
User requesting changes are always kept informed of the status of their change request	38	79.2%
All user requests for changes to the application system must be logged and documented	37	77.1%
All changes to the application program must be logged and documented	32	66.7%
Change requests are classified according to type and importance	29	60.4%
The same routines are used for changes request coming from IS-organization and from the user organization	28	58.3%
All user requests for changes to the application system must be cost justified	26	54.2%
With the exception of emergency fixes, all changes to the application programs are batched for periodic implementation	19	39.6%
Equipment costs associated with operating and maintaining the application system are charged back to the user	19	39.6%
Personnel costs associated with operating and maintaining the application system are charged back to the user	15	31.3%
At acceptance test of changes, checks are also performed to ensure that the documentation is also updated	12	25.0%
A formal audit of the application system is made periodically	4	8.3%

Table 3.9: Use of organizational controls

systems, with a mean of 19%. Some of the comments given by the respondents indicated that CASE had just recently been put to practical use, and that such tools many places only support parts of application development and maintenance.

A similar investigation in England reported that 18% of the respondents were using CASE-tools in 1990 [296] whereas 26% were evaluating the introduction of CASE at that time. In a follow-up study in 1994 [122], 43% of the organizations were currently using CASE-tools. Detailed size-measures for these organizations were not given in [122, 296], thus it is difficult to perform a more detailed comparison. The median number of employees in [122] was between 501 and 2000 [123]. It does not look like any single CASE-tool have a dominant position in Norway, like for instance SDW has had in Holland for several years [184, 331]. No single tool was used by more than two of the Norwegian organizations. A similar situation with no predominant tool is found in the UK [122].

Table 3.11 shows in more detail how the CASE-tools are used. Not surprisingly since almost all CASE-tools contains at least functionality for conceptual modeling [132], this is used by almost all users. Several tools also supported code-generation toward specific supportive software and hardware actors. The percentage of the code of the application system that could be automatically generated varied between 10 and 100% .

3.2.4 Portfolio Analysis

The age distribution of the major application systems is shown in Table 3.12. Average age of the major systems was 4.6 year when regarding the average age of systems older than 10 years to be

Figure	CASE-user			No-user				
	N	Mean	SD	N	Mean	SD	Δ	p
Maintenance total	12	49.4	25.416	34	61.5	23.970	-12.1	.143
Functional maintenance	11	35.4	17.036	33	46.8	17.788	-11.4	.069
Log(employees)	13	3.1626	.786	35	2.4468	.545		**.001**
Log(data department)	13	1.4351	.511	35	.9030	.418		**.002**
Log(system developers)	13	1.0088	.560	32	.6209	.336		**.006**
Log(major systems)	11	.9488	.240	35	.6805	.220		**.001**
Log(user population)	13	2.6572	.540	33	2.2906	.476		**.029**
Organizational controls	12	7.6	2.583	34	5.8	2.583	1.8	**.037**

Table 3.10: Maintenance and size vs. use of CASE

Usage area	Percentage of users
Conceptual modeling (ER, DFD etc.)	92%
Drawing of screens and reports	54%
Storing, administration, and reporting of system information	54%
Code generation	54%
Prototyping/simulation for validation	46%
Generation of DB-schema	46%
Project and process management	31%
Consistency checking of specifications	23%
System test	15%
Reverse engineering	8%

Table 3.11: Use of CASE-tools

15 years[1]. We note that over half (51%) of the systems is less than 3 years old, which indicates that either have much functionality just recently been addressed by computerized support or the replacement rate of system is high.

Looking more closely on the (average) age of the portfolios, we have the following descriptive data:

Number of responses: 49, Range [1.5-12.9], Mean 5.146, Median 4.889.

The average age in the Swanson/Beath investigation was 6.6 years, having many more old systems.

59% of the current portfolio were developed by the CIS-organization, whereas 11% were packages with large adjustments, and 17% were packages with small adjustments. 12% were developed by an outside firm, and the last percent of application systems were developed in the user-organization.

[1]This figure is based on figures of ages of application systems from other investigations [301].

Age of system	# systems	Percentage of systems
0-1 years	54	13
1-3 years	163	38
3-6 years	93	22
6-10 years	76	18
More than 10 years	39	9
Total	425	100

Table 3.12: Age distribution of major systems

In Bergersen, 58% of the projects reported upon were own developments, 27% were packages with large adjustments and 8% were packages with small adjustments (7% were a combination of two of these three).

In Swanson/Beath, 82% of the application systems were developed by the CIS-organization, whereas 15% were developed by outside firms. Only 2% were packages with small adjustments and 1% were developed in the user-organization.

The most notable difference between the two Norwegian investigations and the Swanson/Beath investigation is in the number of packaged systems being used. The greater percentage of packages is probably due to that our organizations are on the average smaller, and as such do not find it cost-efficient in all cases to develop customized solutions. (Swanson/Beath reported a median of 102 persons working in the data department, range [7-266], mean 95 from their case-studies). Another possible explanation is that there exist better packages that are customizable now than it did in the late eighties. On the other hand did we not find any significant correlations between the dominant development manner and the age distribution of the portfolio to support this. The correlations between the percentage of the portfolio that is developed using the different development methods and the size-measures are given in Table 3.13. The Spearman rank is used. From this it appears that organizations with larger data departments, systems developers and number of end-users have a larger percentage of customized systems, whereas larger organizations with large data departments have a smaller proportion of application systems made by external organizations. The large percentage of packages with small adjustments are more often found in organizations with small IS-departments and few end-users.

74.1% of the major systems of the portfolios relied on other systems for their input data. The comparable statistics from Swanson/Beath was 56%. In 41% of the organizations, *all* major systems depended in some sense on data from other systems. The organizations usually supported several technical configurations (range [1-7], mean 2.6, median 2.5). The number of different programming languages being used also ranged from 1 to 7 (mean 2.7, median 2).

Figure	**Percentage developed by IS-organization**		
	r	N	p
Number of employees	.2265	50	.114
Size of data department	**.6016**	50	**.000**
Number of system developers	**.6968**	50	**.000**
Number of major system	.2498	50	.080
Number of end-users	**.4588**	48	**.001**
Figure	**Percentage developed by outside firm**		
	r	N	p
Number of employees	**-.2505**	50	.079
Size of data department	**-.3222**	50	**.023**
Number of system developers	**-.3165**	50	**.025**
Number of major system	.1381	50	.339
Number of end-users	-.1908	48	.194
Figure	**Percentage package, large internal adjustments**		
	r	N	p
Number of employees	.2398	50	.093
Size of data department	.2237	50	.118
Number of system developers	.1577	50	.274
Number of major system	.1619	50	.261
Number of end-users	.0251	48	.865
Figure	**Percentage package, small internal adjustments**		
	r	N	p
Number of employees	-.1783	50	.215
Size of data department	**-.3250**	50	**.021**
Number of system developers	**-.3313**	50	**.019**
Number of major system	**-.3935**	50	**.005**
Number of end-users	**-.2661**	48	**.067**

Table 3.13: Development method vs. size

When looking upon the use of different programming languages, the result of Table 3.14 appeared.

Language	Reported use (45 responses)		Number of systems (30 responses)	
	#	%	#	%
COBOL	33	73	166	49
Diff.4GL	29	64	83	24
C	12	27	15	4
RPG	12	27	15	4
Fortran	7	16	14	4
Assembler	6	13	11	3
PL/1	3	7	8	2
Pascal	2	4	6	2
Other	8	18	22	6
Total			340	

Table 3.14: Use of programming languages

It is interesting to note that none of the object-oriented or object based languages such as C++, Smalltalk, Ada, and Simula were mentioned, even though the figures for C might also include C++ application. Neither any applications in '5GL'-s such as LISP and Prolog were reported.

In Swanson/Beath, most organizations used single languages such as COBOL or PL/1, supplemented with a forth-generation or retrieval language. Based on similar figures from Lientz/Swanson and Nosek/Palvia we conclude that COBOL is still widely used. On the other hand, the use of different 4GLs and partly C appears to be increasing. Based on the figures from the individual systems, those applying a 4GL almost in all cases also had to use a 3GL for parts of the system.

3.2.5 Replacement Systems

100 new systems were currently being developed in the organizations. 48 of these systems (48%) were regarded as replacement systems. In Swanson/Beath, 58 of 117 reported new systems (49%) were replacement systems. 11.4% of the current portfolio were currently being replaced. The similar number from Swanson/Beath was 10% . The average age of systems to be replaced was 8.5 years. The average age for all major systems in these organizations was 4.7 years. This can be compared with the averages for this figure reported in [299] being almost 12 years and [301] being 10 and 8.8 years in two different investigations.

Table 3.15 summarizes the reasons for the replacements being reported. The number in parenthesis indicates how many regarded the factor as extremely important.

Based on this, it seems that the important reasons for replacement are partly that the application systems are hard to maintain, partly a need for moving on to new platforms to be able to standardize or integrate the application systems in the organization. Cautious interpretation of this indicates that troubles with maintaining the system is seen as a main factor in both investigations, whereas the problem of using the old system is perceived as smaller in our investigation.

Replacement reason	Own	Swanson/ Beath
Integration of systems	3.9 (8)	
Burden to maintain	3.7 (13)	3.8
HW/SW changes	3.7 (10)	
Burden to use	3.0 (5)	3.8
Standardization	3.0 (4)	
Burden to operate	2.6 (2)	2.8
Package alternative	2.4 (2)	1.9
Generator alternative	1.8 (0)	1.3
Marking		
5: Extreme importance, 4: Substantial importance,		
3: Moderate importance 2: Slight importance, 1: No importance		

Table 3.15: Reasons for replacement

The other motivations for replacement asked for in Swanson/Beath follow a similar pattern as we find in our investigation.

The average age of the application systems to be replaced was higher than the average experience in the organizations. This might influence the pattern of reuse. On a scale from 1 to 7 where 1 indicates none, 4 indicates 50% and 7 indicates all, the mean amount of reuse of code was 1.75 (median 1). The mean amount of reuse of specifications and design was 3.0 (median 2.0). Thus, when developing replacement systems, reuse of specifications and design appears to be more prominent than reuse of code. Using a Wilcoxon test, we found this to be significant with $p < .01$.

3.2.6 Perceived Problems in Maintenance

A comparison of our results in this area with those of Lientz/Swanson and Nosek/Palvia is given in Table 3.16. Assesments were given on a scale from 1 to 5, 1 indicating no problem and 5 major problem.

The delta values show the difference between our average and those of Nosek/Palvia, and shows generally that almost all problems are perceived to be larger. The most notable exception is *(d) - User demands for enhancements*. This is in contrast with the analysis performed in [237] where no significant change in the perception of the problems between Nosek/Palvia and Lientz/Swanson were found.

To look upon this in more detail, a factor analysis was performed, much in line with Lientz/Swanson and Nosek/Palvia. As reported in [181] a comparison with Nosek/Palvia indicated the support of a trend of more knowledgeable users than what was reported in the Lientz/Swanson investigation. No further significant results were found using these factors.

The above technique has been criticized on its subjectivity both on how respondents interpret the closed factors, and how they assess their importance [160], and that because of the closedness of the technique other problems not in the list will not be mentioned, even if it is room for suggesting own categories in the form [49, 67]. Whereas the first problem is inevitable for this kind

	Problem area	Own	Lientz/ Swanson	Nosek/ Palvia	Δ
a	Turnover of maintenance personnel	3.45	2.23	2.41	+1.04
b	Quality of documentation	3.50	3.00	3.17	+0.33
c	Changes in hardware and system software	2.82	2.04	2.06	+0.76
d	User demands for enhancements	2.87	3.20	3.29	-0.42
e	Skills of maintenance personnel	3.10	2.08	2.19	+0.91
f	Quality of original application system	3.63	2.59	2.58	+1.05
g	Availability of maintenance personnel	3.13	2.58	2.65	+0.48
h	Competing demands for maintenance personnel	3.21	3.03	3.17	+0.04
i	Lack of user interest in application system	2.81	1.87	1.83	+0.98
j	Application system run failures	2.50	1.87	1.82	+0.68
k	Lack of user understanding of system	2.94	2.61	2.62	+0.32
l	Storage requirements	1.96	1.98	1.92	+0.04
m	Processing time requirements	2.33	2.55	2.42	-0.09
n	Motivation of maintenance personnel	2.50	1.92	2.10	+0.40
o	Maintenance programming productivity	2.44	2.04	2.25	+0.19
p	Hardware and system software reliability	2.25	1.81	1.77	+0.48
q	Data integrity in application system	2.91	1.90	1.96	+0.95
r	Unrealistic user expectations	2.96	2.55	2.81	+0.15
s	Adherence to programming standards	2.91	2.11	2.25	+0.66
t	Budgetary pressures	2.63	1.98	2.25	+0.38
u	Inadequate training of user personnel	3.16	2.69	2.65	+0.51
v	Turnover in user organization	2.43	2.36	2.39	+0.04
w	Management support of application system	2.47	1.86	1.90	+0.57
	All over mean	2.823	2.297	2.368	+0.455

Table 3.16: Problem areas in maintenance

of analysis due to social construction, the use of factor-analysis is believed to alleviate the second problem. To address the last critique, Chapin and Dekleva have collected similar data using open-ended questions, Dekleva in the form of a Delphi-study with three iterations. Their results are given in Table 3.17 where the overall grouping of problems is presented of the results on 260 responses on the three major problem areas in maintenance and Table 3.18 where problems were ranged on a scale from 1 to 10 by 44 respondents.

Whereas the investigation of Chapin follows much of the same pattern as in the other investigation, the results of Dekleva is more interesting: The problem that was perceived to be most important was the changing of priorities, a problem not mentioned in the closed list. Due to the length of maintenance tasks, new requests frequently come along before the ongoing task is completed. A great deal of time is wasted on stopping and starting maintenance tasks. From an Italian investigation [92] it was reported that 2/3 of the respondents received change requests *daily*.

Group of problems	Frequency	Percentage
Software characteristics	274	48
Personell factors	149	20
Maintenance management	70	9
Environmental factors	64	8
Activities involving software	57	8
User relations	37	5
Distribution of software	17	2
Miscellaneous	5	0

Table 3.17: Assessed problems in the Chapin-investigation [49]

3.3 Chapter Summary

We have in this chapter given a high-level overview of CIS-support in organizations, as it appears through several survey investigations. Some of the main results seems to be:

- The amount of maintenance seems to be stable around 60% both in investigations from the seventies, eighties, and nineties. Functional maintenance, although significantly less than maintenance, still takes up about 45% of the resources used for development and maintenance. Most of the maintenance is adaptive or perfective, and only a small amount of effort goes to perform emergency fixes.
- Most organizations organize development and maintenance within the same department.
- There seems to be a connection between the amount of functional maintenance, the use of development methodology, and the size of the organizations. Larger organization have a larger percentage of customized systems, and a smaller percentage of packages. Application systems are generally developed on a large diversity of platforms, being written in a large diversity of programming languages.
- The impact of CASE-technology is not yet high in Norway.
- Almost half of the new systems being developed are replacement systems. The most important reasons for replacements seem to be application systems being difficult to maintain, changes in supporting software and hardware, and the integration of application system. The portfolios of most organizations are highly integrated as illustrated in the survey.
- User knowledge and demands seems to be a smaller problem in maintenance, whereas the availability of personell seems to be a major problem area in addition to the quality of the existing application system. Changing priorities do also appear to be a big problem for efficient maintenance many places.

We will cautiously apply the above as background information when discussing requirements to a methodology for COIS-support, and when looking at the potential for applying conceptual modeling within such a methodology in the next chapter.

Description of problem	Rating
Changing priorities	6.4
Inadequate testing methods	6.1
Performance measurement difficulties	6.0
System documentation incomplete	6.0
Rapidly changing business environment	5.9
Large backlog	5.7
Contribution measurement difficulties	5.4
Low morale because of low status	5.3
Lack of experienced maintainers	5.3
Lack of maintenance methodology	5.2
Code being complex and unstructured	5.1
Integration of incompatible systems	5.0
Lack of training of maintainers	5.0
Strategic plans	5.0
Understanding business needs	4.9
Lack of management support	4.8
Antiquated technology	4.7
Lack of support for re-engineering	4.4
High turnover	3.8

Table 3.18: Assessed problems in the Dekleva-investigation [67]

Chapter 4

Methodologies for Computerized Information Systems Support

We will in this chapter give an overview and classification of methodologies for CIS-support in organizations. During the approximately 40 years of computerization, a host of different methodologies have been developed. In 1985, Longworth [201] identified more than 300 and today, this number is by no doubt even higher. We will base our survey of this area on a classification of methodological frameworks which will be presented below. After briefly tracking the evolution of CIS-methodologies from the early years of computing till today, different methodological frameworks covering different aspects of the classification will be presented. The following methodologies and frameworks are presented:

- The conventional waterfall model [270].
- The structured life cycle [342].
- Iterative and throwaway prototyping [44].
- Incremental development [62].
- Transformational and operational development [345].
- Method/1 [7].
- The spiral model [29].
- The hierarchical spiral model [140].
- The fountain model [129].
- REBOOT [161].
- CONFORM [43].
- Maintenance as reuse-oriented development [21].
- Multiview [17].
- STEPS [90].

More detailed overviews of some methodologies being based around the use of conceptual modeling are given in Chapter 7.

4.1 Classification of Methodological Frameworks

We have classified methodologies according to the following areas:

- "Weltanschauung": The basic philosophical view underlying the methodology.
- Coverage in process: The tasks that are covered by the methodology.
- Coverage in product: The parts of a CIS-portfolio that are covered by the methodology.

- Reuse of product and process in the methodology.
- Representation of product and process with emphasis on conceptual modeling.

Below, we will define and discuss each area in more detail.

- **"Weltanschauung":**
  FRISCO [95] differentiate between three different views:
  - Objectivistic: "Reality" exists independently of any observer and merely needs to be mapped to an adequate description. For the objectivist, the relationship between reality and models thereof is trivial or obvious.
  - Constructivistic: "Reality" exists independently of any observer, but what each person possess is a restricted mental model only. For the constructivist, the relationship between "reality" and models of this reality are subject to negotiations among the community of observers and may be adapted from time to time.
  - Mentalistic: To talk about "reality" as such does not make sense because we can only form mental constructions of our perceptions. For the mentalist, what people usually call "reality" as well as its relationship to any model of it is totally dependent on the observer.

  The methodologies that we will present will either be objectivistic or constructivistic. The "weltanschauung" of a methodology is often not explicitly stated in methodology-descriptions, but often appears only indirectly. Since different underlying philosophies may lead to radically different approaches, it is important to establish this. The distinction into objectivistic and constructivistic is parallel to the distinction between objectivistic and subjectivistic in the overview of Hirschheim and Klein [134]. Hirschheim and Klein also distinguish along the order-conflict dimension. In this dimension, the order or integration view emphasizes a social world characterized by order, stability, integration, consensus, and functional coordination. The conflict or coercion view stresses change, conflict, disintegration, and coercion. These two dimensions were originally identified by Burrel and Morgan [42] in the context of organizational and social research.
  Based on the discussion in Chapter 1.1, it should come as no surprise that we find it beneficial to adapt a constructivistic world-view. Note however that we have a somewhat different approach to constructivism than the one described in the FRISCO-report. Both the order and the conflict view combined with constructivism acknowledges a situation of continuous change as discussed in Chapter 1.2. This dimension is not discussed explicitly by us here.
- **Coverage in process:**
  Do the methodology address:
  - Development of application systems
  - Use of application systems
  - Maintenance of application systems

  One or more of the above three areas can be covered, more or less completely and in varying degrees of detail. More detailed specifications of dimensions of development methodologies are given by Blum [26], Davis [60] and Lyytinen [208]. Whereas Davis classifies a methodology according to the way it is able to address varying user-needs over time, Blum classifies development methodologies in two dimensions; if they are product or problem-oriented, and if they are conceptual or formal. We will only look upon the use of conceptual models and if these models are formal or not in the methodologies below. The product vs problem-oriented dimension as discussed by Blum is in our view a distinction on the *part* of development that is covered. Lyytinen includes aspect covered by the "weltanschauung"

and representation of product and process, in addition to linking technical, linguistic, and organizational aspects in a development methodology.

We claim that a comprehensive methodology should cover both development, use, and maintenance in an integrated manner. The emphasis in this thesis will be put on development and maintenance, but also the usage aspect is important, enabling the different end-users to make sense of the existing applications system in the organization, to both be able to use them more efficiently, and to be able to come up with constructive change-request and ideas for more revolutionary changes in the CIS support of the organization when the environment of the organization is changing. We also claim that it is beneficial to not differentiate between development and maintenance in most cases. This is based on the figures appearing in our survey-investigation and in accompanying work.

Maintenance has traditionally been looked upon as a more boring and less challenging task than development [103]. Even if there are indications that this view might be changing e.g. [186] this still appears to be the prominent view among practitioners. According to our discussion in Chapter 1.2, it is both natural and desirable for CISs to change. As shown both in our own and other surveys approximately half of the work which is normally termed maintenance is in fact further development of the information systems portfolio, and should be given credit as such. On the other hand, almost half of the new systems being developed are replacement systems, being functional maintenance not enhancing the functional coverage of the portfolio. Thus seen from the end-users point of view, a better assessment of information system support efficiency seems to be found by blurring the old temporal distinction between maintenance and development, and instead focus on the percentage of functional development. This is difficult to achieve when having a large mental and organizational gap between development and maintenance, even though the actual tasks being done have many similarities.

Swanson [299] recognizes the similarities of the tasks of development and maintenance, but still argues for keeping the old distinction based on the following perceived differences:

- As also noted by Glass [103], a large proportion of traditional maintenance work is to perform un-design of existing systems, finding out what the system does. We will argue that with modern development approaches where as much as possible of the work should take place on a specification and design level, the difference will be smaller. We also note that because of the large amount of replacement work of often poorly documented application systems, code understanding problems are often just as important when developing "new" systems as when maintaining old systems today. Code and design understanding will also often be a problem when reusing the products from other projects, and during traditional development, when due to changing work load, developers have to work on other peoples code for instance during system-test, or because developers are transfered to other projects.
- It is generally believed that "Maintenance of systems is characterized by problems of unpredictable urgency and significant consequent fire-fighting. In difference to new systems development, which is buffered from the day to day tasks of the users, the systems in production is much more visible" [299]. Traditionally, it has been found that approximately 20% of the maintenance work is corrective maintenance [193], and our result of 26% seems to build up on the importance of this. On the other hand, if we look upon the percentage of work that is performed to do immediately necessary corrective maintenance on the application level, we found in our investigation a percentage of 6%. As discussed in Chapter 3 the amount of corrective maintenance appears to

be over-rated by management. In a case study on maintenance in American and Australian organizations of COBOL applications, corrective maintenance was reported to constitute a minor problem where less than 10% of the programs studied had undergone more than two corrective maintenance activities during their lifetime [319]. The significant factor were program complexity and programming style. Related to this is that the results of a survey reported on in [68] which gave no conclusive evidence that organizations using modern development methods used less time on maintenance activities. On the other hand, time spent on emergency error corrections as well as the number of system failures decrease significantly with the use of modern development methods. Systems developed with modern methodologies seemed to facilitate making greater changes in functionality as the systems aged, and the request from users seemed more reasonable, based on a more complete understanding of the system.

The problem of many small maintenance tasks done more or less continuously seems to be increased by how maintenance is often done, in an event-driven manner. In the Jørgensen investigation, where 38% of the tasks were of an corrective nature, as much as 2/3 of the tasks where classified to have high importance by the maintainers themselves. The problem of changing priorities as described by Dekleva [67] is closely related to this.

Even if the problem of emergency fixes seems to be smaller than earlier perceived, a methodology uniting development and maintenance must take into account that one has to be able to perform rapid changes to software artifacts.

Chapin [50] also investigated into this area and concluded that no differences in the nature and characteristics of the demand for or performance of software maintenance work was found between the two types of departmentalization. What those organizations adopting a separate organizational place for software maintenance had achieved was fewer management problems and a more positive attitude toward software maintenance by those managing it, something we believe that also can be achieved by not differentiating between development and maintenance projects in the first place.

- **Coverage in product:**
  Is the method concerned with the development, use, and maintenance of
  - One single application system.
  - A family of application systems.
  - The whole portfolio of application systems in an organization.

  We will argue that it is beneficial for a methodology to consider the whole portfolio and not only the single application system. For the end-users, it is not important which application system that is changed. What is important is that their perceived needs are supported by the complete portfolio.

  Application systems are not developed in a vacuum. They are related to old systems, by inheriting data and functionality, and they are integrated to other systems by data, control, presentation philosophy, and process [308]. As reported in the investigation, the most important reason for replacements apart from systems being unmaintainable, was integration of application systems. Often when doing this kind of integration, it can be useful to collect the functionality of several existing application systems into a new application system, something which is not well supported when having strict borders for what is inside and outside of an application system.

  As noted in [299] the CISs of an organization tend to congregate and develop as families. By original design or not, they come to rely upon each other for their data. In Swan-

son/Beath 56% of the systems where connected to other systems through data integration. In our survey, we found that 73% of the main information systems in the organizations surveyed were dependent on data produced by other systems. In 40% of the responses to this question *all* the main system which the organizations depend upon on a daily basis were dependant on data produced by other systems.
Over time, newer application systems originate in niches provided by older ones, and identifiable families of systems come to exist. Relationships among families are further established. In the long run, an organization is served more by its CISs as a whole than it is by the application systems taken individually.

- **Reuse of product and process:**
Reusing experience is a key ingredient to progress in any discipline. Without reuse everything must be re-learned and recreated; progress in an economical fashion is unlikely. The need to utilize extensive reuse is based on the need for rapid changes in the CIS of an organization as discussed in Chapter 1.2.
An overview of dimensions of reuse is given by Prieto-Diaz [263]:

  - By substance: The essence of the items to be reused:
    - Idea reuse involves reusing formal notions, such as a general solution to a class of problems.
    - Artifacts reuse: The artifacts can be code, conceptual models, design, specifications, objects, text, architectures, or test data.
    - Procedures reuse: Formalizing and encapsulating software development procedure. Procedures reuse also means reusing skills and know-how, i.e. having a development and maintenance methodology can be looked upon as reuse in this sense.
  - By scope: The form and extent of reuse:
    - Vertical reuse is reuse within the same application area.
    - Horizontal reuse is reuse across application areas.
  - By mode: How reuse is conducted:
    - Planned reuse: The systematic and formal practice of reuse, guidelines and procedures for reuse have been defined, and metrics are being collected to assess reuse performance.
    - Ad-hoc reuse: An informal practice, in which components are selected from general libraries.
  - By technique: How reuse is implemented:
    - Compositional reuse is the use of existing artifacts as building blocks for new systems.
    - Generative reuse is reuse at the specification level by means of design and code-generators.
  - By intention: Defines how elements will be reused:
    - As-is or black-box reuse is reuse without modifications.
    - Modified or white-box reuse involves modifications of what is reused.

  It is usual to differentiate between methodologies being *for reuse* and those being *with reuse* [161, 332]. Another distinction is between reuse-in-the-large and reuse-in the-small. We will restrict the use of the term in the evaluations to include the planned reuse of artifacts, i.e. not including that using a methodology is an example of procedures reuse.
- **Representation of product and process:**

Knowledge about the process and the product of CIS development and maintenance can be represented using different kinds of languages. These languages can be informal, semi-formal, or formal, having a mathematical and/or a executional semantics as defined in Chapter 2.1.

We will in this thesis concentrate on conceptual modeling languages. As will be illustrated, conceptual modeling is believed to be an important technique for CIS support in organizations when combining development and maintenance having support for not only a single application systems, but the whole application system portfolio, being based around social construction theory and reuse. When discussing the benefits of using conceptual modeling below, we should have in mind that we are primarily talking about partly graphical languages which are semi-formal or formal, have a limited vocabulary, and which can be used in many areas on varying levels of formality and completeness.

- A conceptual model has the possibility of being a problem-oriented description of the requirements for CIS support, without being restrained too early by technical constraints. In this way we believe one can more easily support a process of social construction of information systems. A problem-oriented approach has been asked for by many researchers [32, 38, 119, 313] and conceptual modeling is looked upon as one way of achieving this, even if certain approaches to conceptual modeling such as structured analysis and object-oriented analysis have been criticized for being too target-oriented by many [117, 119, 137]:
  - They focus too early on design issues such as procedural decomposition [39] or object interaction [264],
  - They do not capture requirements in the form of e.g. business rules [313] directly enough.
- Due to the visual nature of many conceptual modeling languages they are believed to be more helpful in the sense-making process of what is modeled than the model which is implicit in the code of an application system. On the other hand, if we want to refine the conceptual models into a form that is suitable for automatic code-generation, the essential difficulty of complexity [35] will again appear, thus filtering techniques [282] are necessary.
- Since the separate conceptual modeling languages only include a limited set of phenomena, this enable a focusing of concerns, and it is possible to deduce properties that are difficult if not impossible to perceive directly, by concentrating on only some aspect at the time. This is obviously also problematic if this makes one blind for other concern, or makes it impossible to externalize certain explicit knowledge. Based on this we will claim that one need a set of interrelated semi-formal and formal modeling languages which can cover different perspectives for conceptual modeling to be more generally useful.
- Conceptual models developed in early parts of development can be used as an outset for further design and implementation, supporting generative reuse. Conceptual models are also believed to be easier to maintain than textual documents that do not have any other mission than to serve as documentation, since they can be constructed as part of the process of developing and maintaining the application system in the first place, thus supporting change and an integration of development and maintenance techniques. It is also easier to get an overview of the CIS-support of an organization if the languages for conceptual modeling are known and sufficient tool support for handling them exist, thus potentially supporting the evolution of the whole portfolio.

As noted in the survey, most of the organizations having started to use CASE-tools for development and maintenance, use these for conceptual modeling. We will return in Chapter 5 with a more thorough description of this area.

Taking a philosophical standpoint neither reuse nor conceptual modeling nor having a defined methodology can be optimal, since all situations are unique, and thus in principle can best be attacked by using unique means. Reusing artifacts originally produced for some other purpose, in effect means to apply an externalization of the local reality of someone else than the current stakeholders, which thus can not be optimal. On the other hand, reuse is performed all the time. Using a commercial DBMS is for instance reuse, but it is not very wise to produce your own database management system when you perceive a need for this kind of functionality if you do not have very special needs. Said bluntly, it is not very useful to use ten years to develop an application system, because one wants an optimal process of social construction. A balance between the different concerns brought up by our philosophical outlook is thus necessary.

4.2 A Survey of Methodological Frameworks

In the early days of computing, the main technique was code-and-test. In 1968, the first of two NATO conferences on Software Engineering was held. With many of the programming problems mastered, it was time to confront the difficulties encountered in developing application systems. In the rest of the sixties and the seventies, several important principles where established, such as abstraction-levels (Dijkstra), stepwise refinement and functional decomposition (Wirth), structured programming (Dijkstra) and design including coupling and cohesion measures (Stevens, Myer, and Constantine), and information hiding (Parnas).

For systems development as a whole, so-called life cycle models were introduced. According to Yourdon [342] their purpose is threefold:

- They define the activities to be carried out.
- They provide checkpoints for management control.
- They introduce consistency among projects in the same organization.

4.2.1 The Waterfall Model

The waterfall model, illustrated in Figure 4.1, is usually attributed to Royce [270]. It organizes CIS projects as a linear sequence of phases, where each phase is completed by documenting its achievements. In addition, there are feedback loops between successive phases that enable the modification of the document from the previous phase.

The perceived benefits of using the waterfall model can be summarized as follows [62]:

- It instructs the developers to specify the system prior to the construction of it.
- It encourages one to design the system components before they are actually coded.
- By viewing the project as a sequence of phases, the managers can more easily control the progress of the work, and use the defined milestones as a tool for deciding if the project is to continue or not. It will also help the managers to set up a structured and manageable project-organization.
- One is required to write documents that will ease the testing and documentation of the system, and that will reduce maintenance costs.

Every project starts out with a feasibility study. The main problems are identified and the pursuit of a new CIS is justified in terms of unfulfilled needs and wishes.

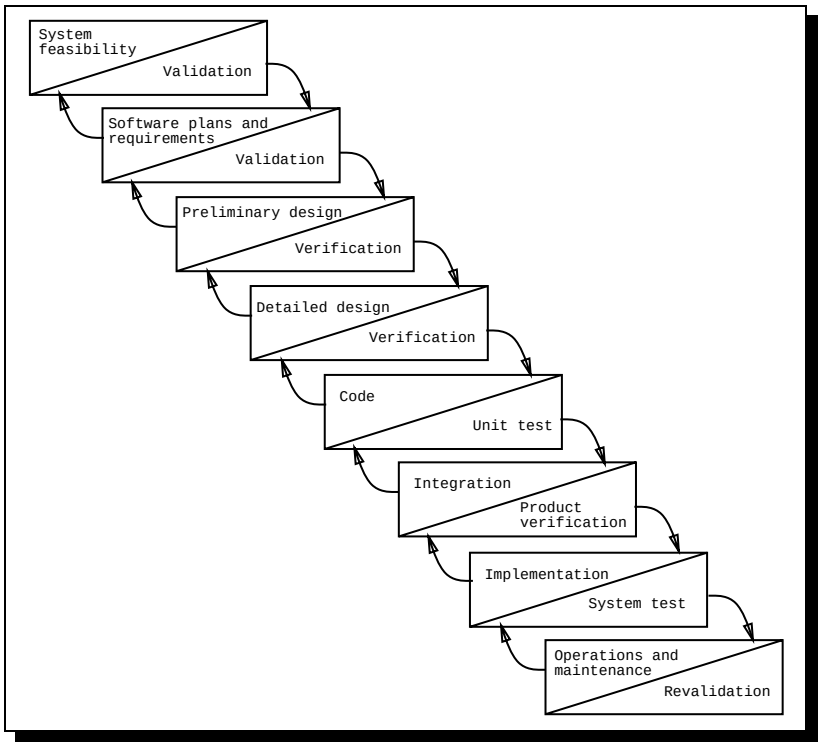

Figure 4.1: The waterfall model of system development (From [28])

The purpose of the requirements specification is to define and document all the stakeholder's needs. The specification is supposed to contain a complete description of what the system will do from a user's point of view. How the system will do it, is deliberately ignored. This is addressed during design.

Most "standard" methodologies for commercial organizations, government contractors, and governmental organizations e.g. ISO9000 [145], DOD-STD-2167A [70], and SSADM [76] follows some basic variation of this model, even if the number of phases and the names of the phases often varies [60].

The (conventional) waterfall model has received much criticism since the beginning of the eighties [218, 300]:

- The phases are artificial constructs, "one specific kind of project management strategy imposed on software development" [218]. It is in practice often difficult to separate specification completely from design and implementation [300].
- An executing system is presented first at the end of the project. This is unfortunate of several reasons:
  - Errors made in the specification will be more difficult and thus more costly to remove when they are not discovered before the end of the project.
  - The customer and end-users may get impatient and press for premature results or lose interest in the project because they do not see any result of the work that has been done.
  - The communication gap between the users and the developers arising because of their different realities is not attacked.
- Systems developed using the conventional methodology are difficult to change, resulting in poor support for system evolution.

Classification of the conventional waterfall model

- Weltanschauung: Usually objectivistic even if this is not always stated explicitly.
- Coverage in process: Primarily focused on development.
- Coverage in product: A single application system.
- Reuse: Not explicitly supported.
- Conceptual modeling: Only applied shallowly if at all.

4.2.2 Alternatives and Extensions of the Waterfall Model

We will here present some of the methodologies that have grown out of extensions and critique of the waterfall-methodology before looking closer at newer methodological frameworks.

The structured life cycle

In structured analysis [342] a set of graphical documentation techniques are employed to specify the functional requirements in a top-down manner:

- Data flow diagrams [97] document the overall functional properties of the system.
- Entity relationship diagrams [53] or models in a similar semantical data modeling language model entities and the relationship between these entities.
- A data dictionary is used to record definitions of data elements.
- State transition diagrams may be used to specify the time-dependent behavior (control structures) of the system.
- Process specifications can be written in a variety of ways: decision tables, flowcharts, graphs, "pre" and "post" conditions, and structured English.

Structured design is defined as "the determination of which modules, interconnected in which way, will best solve some well-defined problem" [342]. It is assumed that structured design has been preceded by structured analysis. The design process is guided by design evaluation criteria and design heuristics, resulting in a set of structure charts.

Many of the same critical remarks have been made towards this methodology as to the conventional waterfall methodology, and in our classification scheme, it basically differs from this by its emphasis on conceptual modeling.

Classification of the structured life-cycle

- Weltanschauung: Objectivistic.
- Coverage in process: Primarily focused on development.
- Coverage in product: A single application system.
- Reuse: Not explicitly supported.
- Conceptual modeling: Emphasis on the use of semi-formal conceptual modeling languages.

Prototyping

A prototype can be defined as "an executable model of (parts of) an information system, which emphasizes specific aspects of that system" [322].

Prototyping as *technique* is usually seen as a supplement to conventional application system development methodologies [303, 322]. It put emphasis on high user participation and tangible

representations of user requirements. The iterative generation and validation of executable models makes the approach particularly useful when the requirements are unstable or uncertain.

Prototyping as *methodology* is a highly iterative process, which is characterized by extensive use of prototypes [44, 322]. The objective is to clarify certain characteristics of an application system by constructing an artifact that can be executed. On the basis of user feedback the prototype is revised and new knowledge and new insights are gained. After a series of revisions, the prototype is acceptable to the users, which indicates that it reflects the user requirements in some specific aspects.

According to Vonk [322], prototyping differs from a traditional methodology on the following areas:

- The users can validate the requirements by testing a corresponding executable model. The communication between users and developers is improved in that users can directly experience the consequences of the specified requirements.
- Traditional tool-support for these methodologies have been unable to focus on the user interface aspects. Prototypes exploit the execution of (simplified) user interfaces to improve the externalization of user requirements.
- The requirements tend to change as the project is carried out. Instability of functional requirements is easily handled through the interactive generation and validation of functional prototypes.

In Vonk's opinion, its main benefit is the reduction of uncertainty. The choice of development strategy, thus, should be based on the judgement of project uncertainty. Carey [44] sees the following advantages with prototyping: Faster development time, easier end-use and learning, less manpower to develop systems, decreased backlogs, and enhanced user/analyst communication. Some disadvantages include: The fostering of undue expectations on the part of the users, what the users sees may not be what the users gets, and the availability of application generator software may encourage unduly end-user computing.

Various prototyping methodologies: There are two main types of prototyping [44, 322]: In iterative prototyping, also called evolutionary prototyping, the prototype evolves into the final application system after a series of evolutionary user-initiated changes. In throwaway prototyping the prototype is only used to help to establish the user's requirements to the application system. As soon as the process is finished, the prototype is discarded and a the real application system is implemented.

The two different life-cycles are illustrated in Figure 4.2.

Classification of prototyping

- Weltanschauung: Usually objectivistic, but not necessarily so. Many methodologies having a constructivistic world-view applies prototyping actively as a technique.
- Coverage in process: Primarily focused on development, especially on early phases of development.
- Coverage in product: A single application system.
- Reuse: Not explicitly supported, but will often be applied in rapid throwaway prototyping in some form (development with reuse).
- Conceptual modeling: Not mandatory, but can be used as a starting point for functional prototyping if the conceptual modeling languages used have a defined operational semantics.

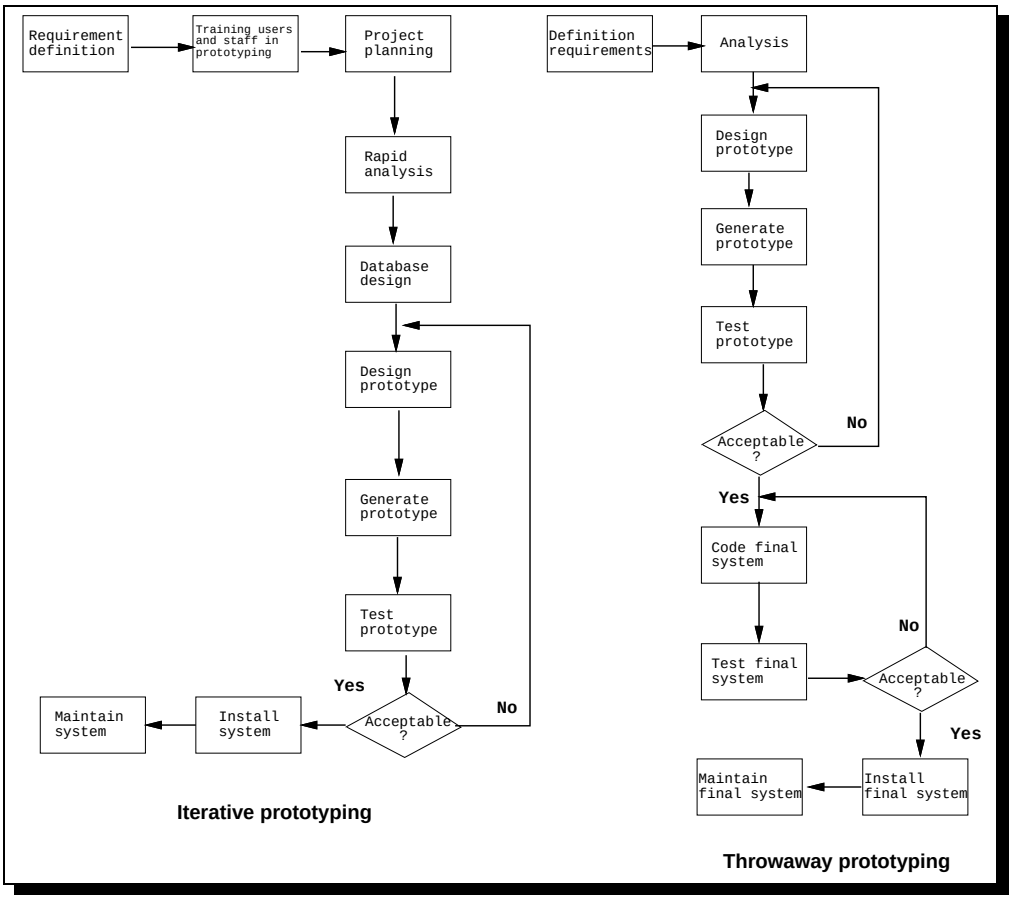

Figure 4.2: Iterative vs. throwaway prototyping (From [44])

Incremental development

Incremental development [62] is the process of constructing a partial implementation of a total system and slowly adding increased functionality or performance. When the increments are released to the production environment one by one, one often talks about stage-wise development. This is meant to reduce the costs incurred before an initial capability is achieved. It also produces an operational system more quickly, and it thus reduces the possibility that the user needs will change during development. Incremental development presupposes that most of the requirements are understood up front, and one chooses to implement only a subset at a time. Note how this differs from evolutionary prototyping, even if these techniques could be integrated. Since it is possible to use an incremental strategy together with methodologies that are different according to our classification, a separate classification of incremental development is not given.

Transformational and operational development

The transformational approach which is also often termed automatic software synthesis [204] assumes the existence of formal specification languages and tools for automatic transformations.

As shown in Figure 4.3a, the formal specification is the crucial element. It is used as a starting point for the transformation process, but is at the same time the main document for stakeholder validation. This illustrates the main problem using transformational techniques: On the one hand

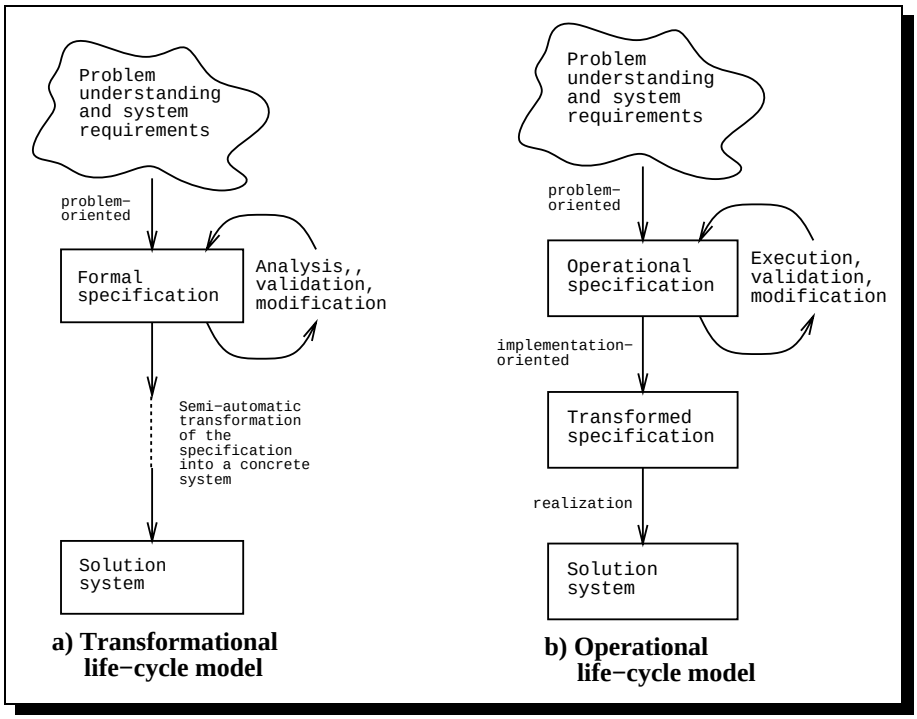

Figure 4.3: The transformational and operational life cycle models

formality is necessary to apply automatic transformations, on the other hand it makes specifications more difficult to understand for end-users.

A series of transformations are applied to reach the final target code. During these transformations, additional details are added to the specification. Not all of these details can be automatically added, and a developer is needed to guide the transformation process. The sequence of transformations and decisions is kept in a development record. Using this record, one can maintain and re-implement the formal specification.

The operational approach is described in detail by Zave [346]. Its main characteristics are (1) the separation of problem-oriented and implementation-oriented system features, and (2) the provision of executable system models early in the development process. It is claimed that the approach will enhance the validation process as prototypes are immediately available. The approach is illustrated in Figure 4.3b. The approach rests on the use of an operational specification language. This language is defined as a suitable interpreter making the models available through prototyping or symbolic evaluation or both [125] for validation.

The specification model is rarely suited as the final program code. It is a functional model, although ideas to include nonfunctional requirements have been explored (e.g. [345]). Resource management and resource allocation strategies are usually omitted, and characteristics of the target environment are deliberately ignored. As stated by Agresti [1], the operational paradigm violates the traditional distinction between “what” and “how” considerations. Instead the development process is separated on the basis of problem-oriented versus product-oriented concerns.

As soon as the operational model is finished, a series of transformations are carried out. The goal of these transformations is to reach another specification which is directly interpretable by the target processor. In order to do so, decisions concerning performance and implementation resources are made.

The approach is claimed to have several advantages to conventional waterfall life cycle models [345]:

- It exploits the advantages of formality.
- Rapid prototypes or symbolically executable units are available all away from the start.
- Since the transformations preserve correctness, it is not necessary to verify the final code.
- All functional modifications are done at the specification level.

Also, in Zave's opinion the separation of problem-oriented and implementation-oriented issues is useful to improve the system's maintainability. Operational specifications are constructed with maintenance in mind, while transformations try to take care of requirements concerning performance and efficiency. Conventional techniques, on the other hand, supports only one decomposition principle. The conflicting issue of efficiency and ease of maintenance must be addressed in the same process, which tends to result in more or less unconscious compromises.

Among the disadvantages are the danger for premature design decisions, the difficulties of comprehending the specifications, and the problems of guiding the transformations.

Classification of operational and transformational approaches

- Weltanschauung: Usually objectivistic.
- Coverage in process: Focused on development, although claiming improved maintainability.
- Coverage in product: A single application system.
- Reuse: Based on generative reuse. Compositional reuse is not specifically supported.
- Conceptual modeling: Formal conceptual modeling languages such as those found in PAISLEY [345], GIST [20], and STATEMATE [126] are usually the cornerstone of the operational approach.

4.2.3 Method/1

Some of the main aspects discussed above is illustrated in the commercially used methodology METHOD/1. The methodology which is used by Andersen Consulting includes four different life-cycle-models, two for custom systems development covering both small and large projects both for host and client/server architectures, one for package systems development and one for rapid development [8] applying an incremental strategy. The methodology includes the description of a plethora of tasks, and project management will in the beginning of a project select the tasks judged to be necessary for a successful project. It is also possible to specify additional tasks and documents. The tasks will have defined inputs and outputs, and as such interdependencies between tasks are described. Based on this information the project plan is developed. Whereas the rapid development methodology is an example of iterative incremental prototyping, the methodologies for custom systems development are most used, and these are briefly described below. The overall phases are shown in the top of Figure 4.4, which are taken from an earlier version of the methodology.

Comparing with Figure 4.1, the first phase in METHOD/1 is not covered in the traditional waterfall life cycle, systems design correspond to the first four phases, systems installation with the next two phases and production systems support with operation and maintenance. A more detailed overview of systems design is given in the lower part of Figure 4.4.

The figure shows the overall pattern, without going into details on the deliverables and inputs and outputs. In the bottom of Figure 4.4 , the 'hardware and system software direction' task

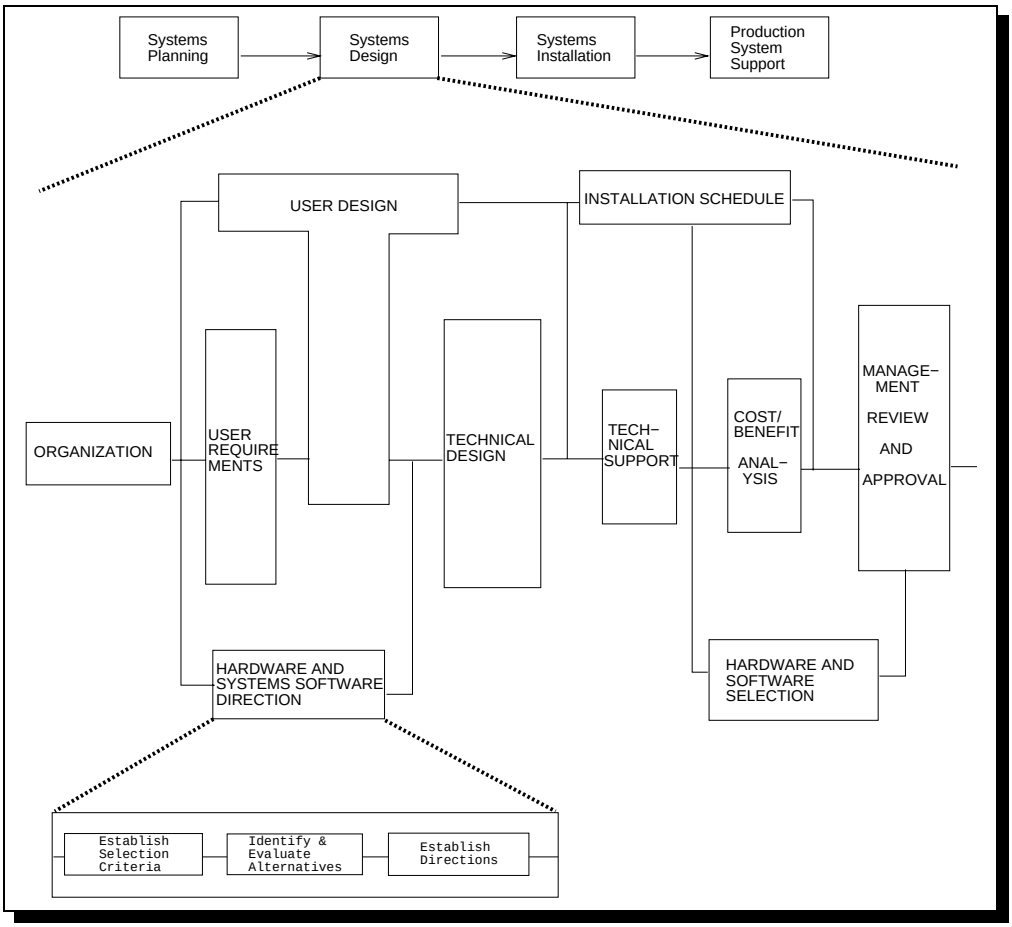

Figure 4.4: Method/1 for customized systems

is exploded into its contained tasks. The user design part of systems design support throwaway prototyping. Conceptual modeling is also supported, but is not given the same emphasis as in the structured life cycle or the operational approach.

Classification of METHOD/1

- Weltanschauung: Objectivistic.
- Coverage in process: Primarily focused on development.
- Coverage in product: Systems planning potentially covers the complete COIS on a high level. The other phases covers a single application system.
- Reuse: Not explicitly supported. Reuse-in-the-large is somewhat supported in the package-development life cycle, not presented above.
- Conceptual modeling: Not mandatory, but use of semi-formal conceptual modeling languages are supported in early systems design.

4.2.4 The Spiral Model

The spiral model was introduced by Boehm in 1988 [29]. It may be perceived as a framework for systems development, in which risk analysis governs the choice of more specific methodologies as

the project progresses. The spiral model potentially subsumes both the prototyping (both iterative and throwaway), operational/transformational, and waterfall methodology.

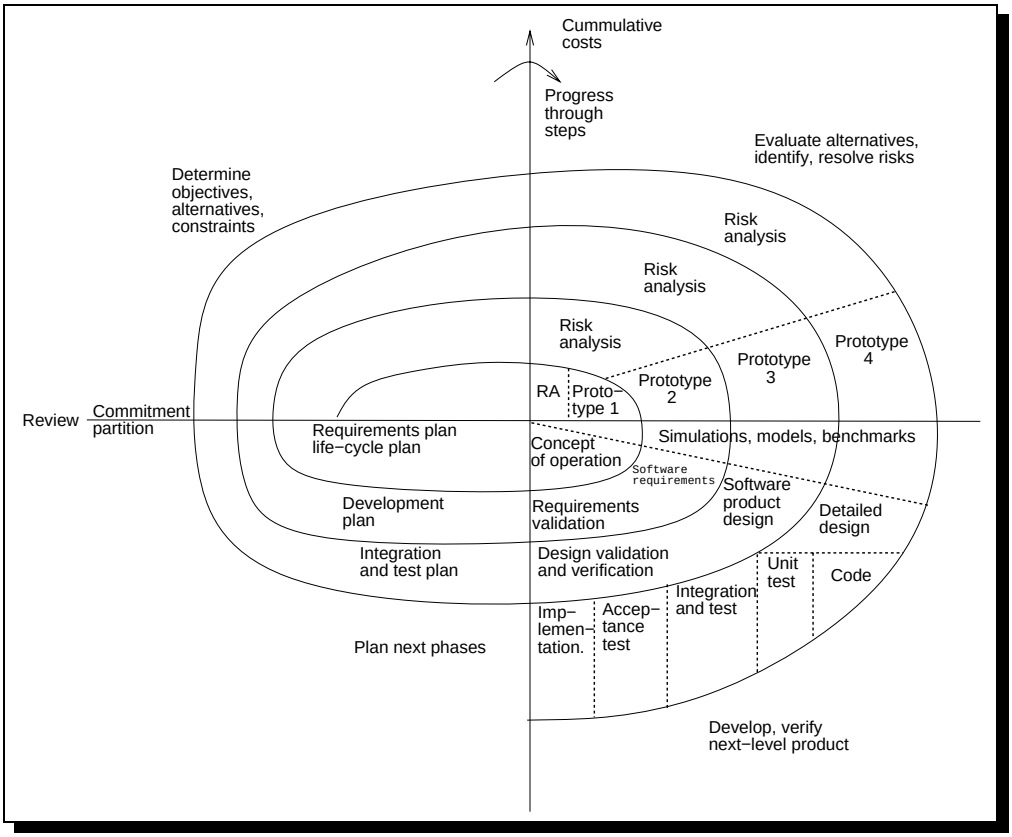

Figure 4.5: The spiral model (From [29])

As shown in Figure 4.5, the project is intended to iterate through a number of basic steps. Each iteration encompasses some objectives to be solved, and is comprised of the following steps:

- Determine objectives, alternatives, and constraints.
- Evaluate the alternatives and identify risks connected to central components or features.
- Develop product to resolve the most critical risks, and evaluate the results. Prototyping, reuse techniques, and requirement and design specifications are all means of reducing risks.
- Plan the next iteration and review the achievements of the current one.

According to Boehm the strengths of the spiral model are:

- It focuses early attention on options involving the reuse of existing software.
- It accommodates for life-cycle evolution, growth, and changes of the software product.
- It provides a mechanism for incorporating software quality objectives into software product development.
- It focuses on eliminating errors and unattractive alternatives early.
- For each of the sources of project activity and resource expenditure, it answers the key question: How much is enough?
- It does not involve separate techniques for software development and maintenance.
- It provides a viable framework for integrated hardware-software systems development.

Its major weakness is connected to the availability of proper risk analysis techniques. As long as risk determination is more an art than an engineering discipline, the model will give a rather the-

oretical and superficial impression. Moreover, the iterated reviewing of current objectives may impose some troubles to the specification of contracts between customers and developers. As presented in [29] it is merely a framework for development and maintenance which will be difficult to apply directly by inexperienced developers.

Classification of Spiral Model

- Weltanschauung: Objectivistic.
- Coverage in process: Both development and maintenance covered by the same framework, but emergency error-correction is not discussed specifically.
- Coverage in product: Development or maintenance of a single application system.
- Reuse: Development with reuse mentioned as one of several risk management techniques.
- Conceptual modeling: Not specifically supported, but the framework is open for its use.

4.2.5 The Hierarchical Spiral Model

The hierarchical spiral model [140, 139], is an extension of the spiral model as described in Figure 4.5. The extensions are primarily based on the work on the PIOCO-methodology [142]. It is based on three levels of modeling an IS:

- **O:** The organizational level, which defines the organizational role and context of the IS.
- **C/I:** The conceptual/infological level, which defines an implementation independent specification for the IS.
- **D/T** The datalogical/technical level, which defines the technical implementation for the CIS.

The major point about the three levels is that they are interpreted as abstractions from three different, even though interdependent domains: the host organization which the CIS is to be embedded in, the UoD about which the CIS is to communicate data, and the technology to be used in the implementation of the application system. The abstractions are described in semi-formal or formal languages, and can be both descriptive and prescriptive. Three aspects are distinguished on each level: Structural, functional, and behavioral.

The methodology is based on two major assumptions: First, that the CIS as the target of design should provide a basic structure for the development process, and, second, that the three major levels of modeling represent such different perspectives on the IS that it is justifiable to structure the development process according to them.

Each main phase in the hierarchical spiral model can be represented as a spiral structure similar to the framework of Boehm.

Each sub-phase starts with the planning of the sub-phase including activities of the following kind.

- Identification of objectives of the next sub-phase in terms of risk resolution and/or refinement of the product.
- Identification of constraints.
- Identification of alternatives.
- Analysis of factors affecting the outcomes and costs of the alternatives.
- Assessment and selection.
- Detailed planning.

The primary design level is followed by a look-ahead analysis (e.g. C/I and D/T design in the O-phase) leading to possible verification and validation of the design artifacts.

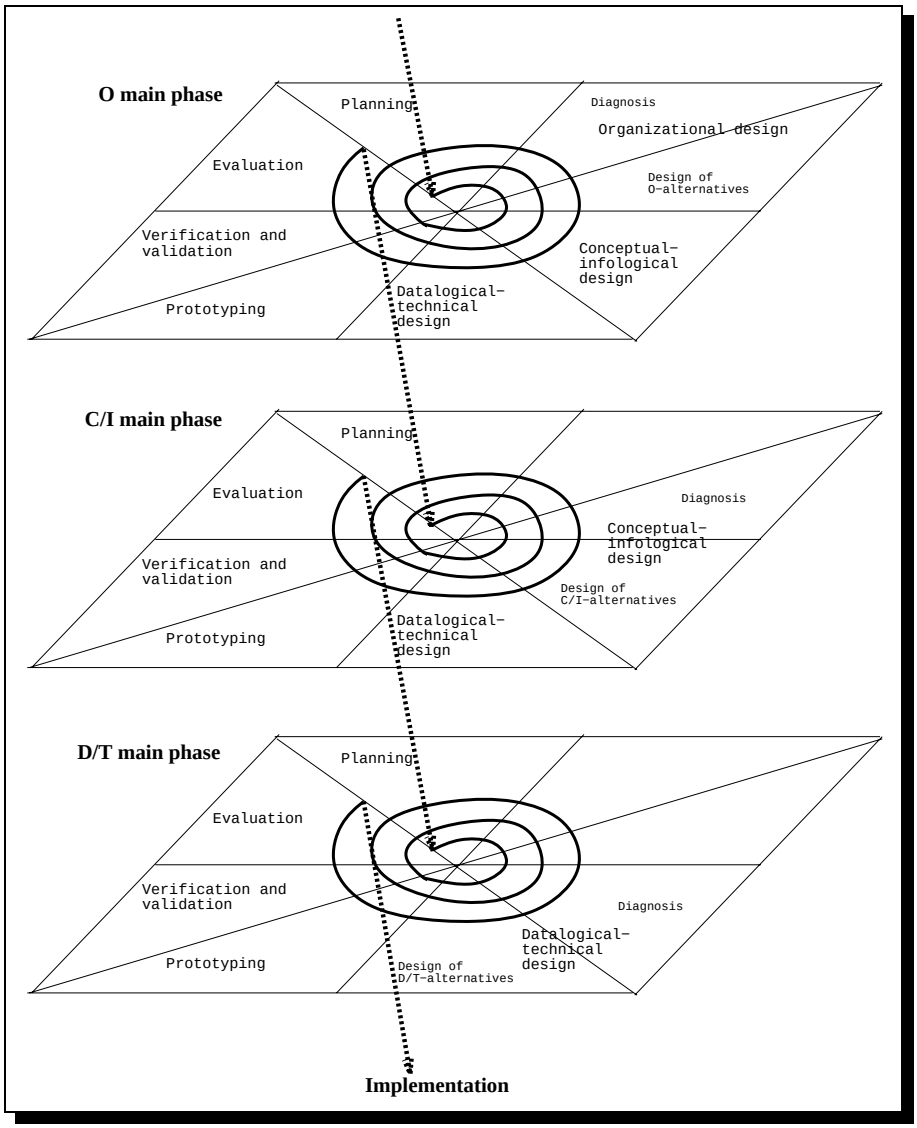

Figure 4.6: The hierarchical spiral model (From [139])

Many of the same advantages and problems as with Boehm's spiral model is claimed. Compared with the spiral model, three areas are improved upon:

- The theoretical basis is extended.
- The approach is based on three levels of modeling information systems. In addition to being risk-driven, the approach is also model-driven.
- The idea of the spiral model as an integrative framework that can incorporate more specific development methodologies such as the waterfall methodology, prototyping and evolutionary development is extended.

Classification of the Hierarchical Spiral Model

- Weltanschauung: Objectivistic.
- Coverage in process: Both development and maintenance is covered by the same framework, but emergency error-correction is not discussed specifically.

- Coverage in product: Although organizational modeling potential can cover larger parts of a portfolio, development and maintenance of a single application system are emphasized.
- Reuse: It is stated that development with reuse is enabled.
- Conceptual modeling: The methodology is based in a large degree on active use of conceptual models, even if no set of concrete modeling languages are described in [140, 139].

4.2.6 Object-oriented Approaches and the Fountain Model

Structural development and especially its emphasis on top-down design has been criticized on the following grounds [222]:

- Top-down design takes no account of evolutionary changes.
- In top-down design, the system is characterized by a single function, something which is questionable.
- Top-down design is based on a functional mind-set, and consequently the structural aspect is often neglected.
- Top-down design does not encourage reusability.

Several researchers have claimed that the object-oriented approach will alleviate the weaknesses of this practice, being perceived to have the following characteristics [205]:

- At every phase of the development process, the system may be analyzed and specified using the same structural phenomena. In addition objects are presumably easier to identify in the problem domain than functions and states.
- Reuse of previously constructed components is encouraged.
- The principles of information hiding and inheritance enhance system maintainability.

As opposed to structured analysis, the object-oriented approach supports bottom-up development, and perhaps also the mixture of bottom-up and top-down development.

According to Slonim [289], there are currently more than 150 suggested approaches to object-oriented analysis and design, with no clear leader. Most approaches have their separate theory, terminology, and modeling approach. Early object-oriented methodologies were usually language-specific, especially geared towards Ada development [129]. Since Ada is not object-oriented, but object-based [326], some of the major features of object-orientations such as inheritance are not covered in these methodologies. The Fountain-model, one of the first comprehensive frameworks for object-orientated development, was presented in [129]. This is divided into the following phases:

- Analyze the system requirements in terms of objects and services.
- Identify and define the objects and their services.
- Establish object interactions on the basis of services required and services rendered.
- Merge analysis and design with the use of lower-level EDFDs [18].
- Investigate library classes for bottom-up development.
- Introduce hierarchical relationships (super-classes and subclasses).
- Undertake aggregation and generalization of classes.

The overall development process is both iterative and integrated, in that the object specification in one phase is a refinement of the specification in the previous phase. The objects are often clustered together, and each cluster is given a separate life cycle. The emphasis on object encapsulation makes it possible to run independent sub-projects on different clusters of the total system.

The fountain-model is illustrated in Figure 4.7. Merging and overlap in activities are represented in the figure by the degree of overlap of the circle symbols. The expected decreasing need for maintenance is illustrated by the smaller circle. Since the foundation of a successful appli-

cation system project is its requirements analysis and specification, this phase has been placed at the base of the diagram. The life-cycle thus grows upward to a pinnacle of use, falling only in terms of necessary maintenance. This effectively reverts the phase of the cycle to a lower level. Applications of this model to specific environments may lead to software life-cycles such as prototyping, and can also incorporate more traditional functional decomposition.

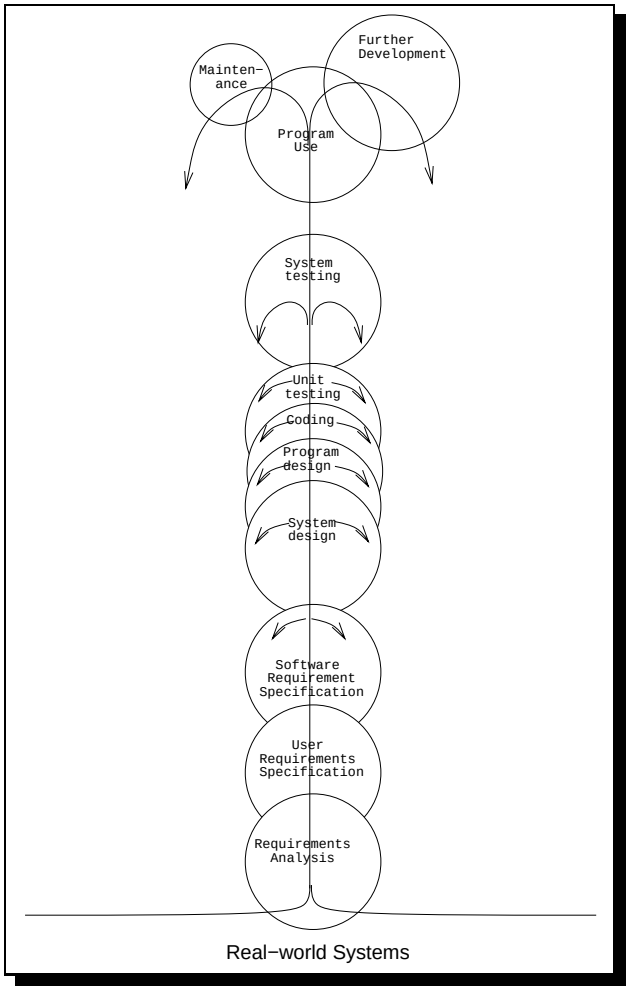

Figure 4.7: The Fountain model (From [129])

The system can also be viewed in terms of a synergistic amalgamation of a large number of autonomous classes. Modifications are claimed to be more easily done interactively between class development and systems specification so there is no longer a need to freeze the overall application systems specification at an early stage.

Classification of Fountain model

- Weltanschauung: Objectivistic.
- Coverage in process: Primarily focused on development, but also maintenance and use are briefly mentioned.
- Coverage in product: A single application system.
- Reuse: Development for (and partly with) reuse.
- Conceptual modeling: Methodology based on object-oriented conceptual modeling.

Organization	Project management	Development process for reuse	Development process with reuse	Library	Metrics
Reuse strategy	External coordination	Development process integration		Artifact data	Product metrics
Reuse assessment	Project planning	Type of produced/reused data		Artifact classification	Process metrics
Legal issues	Project tracking	Variability analysis	Functionality evaluation	Change management	
Cost & pricing	Staffing	Generality expression	Reuse cost evaluation	Library maintenance	
Product offer		Cost/benefit analysis	Adaptation/ integration		

Table 4.1: The REBOOT key reuse areas and factors

4.2.7 REBOOT

REBOOT (Reuse Based on Object-Oriented Techniques) was an ESPRIT-III project which ended in 1994 [161]. REBOOT does not require a shift in development methodology, but rather offers a potential add-on to deal with the reuse-specific activities. It is especially suited to support object-oriented approaches.

The following problems were addressed:

- Organizational aspects of reuse, concerning reuse introduction, organizational and managerial issues, and economic and legal aspects.
- Technical aspects of reuse, concerning both development for and with reuse.
- Reuse introduction: During initiation, the objectives for the reuse initiative are determined. In the second stage, the organization's reuse opportunities are analyzed, and its reuse maturity is assessed. The assessment of the organization's maturity is based upon REBOOT's reuse maturity model, inspired by SEI's CMM [252]. The same five maturity levels are used, i.e. chaotic, repeatable, defined, managed, and optimized, but the assessment is specifically related to reuse capabilities, not the overall capabilities for software production. Five key reuse areas have been defined according to the five parts of the REBOOT methodology, and for each of these several key reuse factors have been identified, as shown in Table 4.1. Each of these factors can be mastered to various degrees by the organization, thus determining its maturity level. The assessment will discourage organizations from adopting technology they are not yet mature for, and it will indicate what steps to take next to improve the reuse capabilities.
- Organizational and managerial issues: The reuse organization can be structured around project, artifacts, or domain.
- Technical aspects: When it comes to the technical solutions for reuse, REBOOT uses a artifact library technique, where reuse is done by retrieving reusable artifacts from the library and integrating them in new applications. REBOOT's ambition was to support the activities directly related to reuse, such as
  - Development for reuse:
    - Re-engineering
    - Qualification: Ensuring that artifacts to be entered into the library are of acceptable quality. In addition to measuring quality, the REBOOT qualification service

    also collects data such as development history and context of use about the artifact.
    - Classification: Classifying the artifact to facilitate subsequent retrieval.
- Development with reuse:
    - Retrieval: Finding artifacts according a specified need.
    - Evaluation: Evaluating the quality and appropriateness of candidate artifacts.
    - Adaptation: Adapting the artifact if it cannot be reused as-is.

REBOOT supports reuse in all phases of development, so artifacts can include analysis models, designs, code, documentation, and test cases. Some of the support assumes that artifacts are object-oriented.

The most important point about REBOOT may be that it has dealt thoroughly both with the technical and organizational aspects of reuse in a comprehensive manner. Thus, an approach to reuse introduction and organizational has been provided together with an accompanying technology to fill the various needs an organization will have as its reuse capabilities increase. Future extensions is the integration of the environment with more CASE-tools. One also see a need for further elaboration of the classification and retrieval approach.

Classification of REBOOT

- Weltanschauung: Objectivistic.
- Coverage in process: Potentially both development and maintenance when combined with an existing methodology.
- Coverage in product: Potential gain for whole portfolio.
- Reuse: Support for development for and with reuse.
- Conceptual modeling: Can be included, especially object-oriented modeling, but the project primarily focused on reuse of detailed design and code.

4.2.8 CONFORM

CONFORM (Configuration Management Formalization for Maintenance) [43] is a method which provide guidelines for carrying out a variety of activities performed during maintenance. The method accommodates a change control framework around which software configuration management (SCM) is applied. The aim is to exert control over an existing application system while simultaneously incrementally re-documenting it. In order to enforce software quality, a change control framework has been established within CONFORM called the software maintenance model (SMM). SMM is illustrated in Figure 4.8.

The rectangles in the figure represent SMM phases and the ovals represent the baselines formed from the output of the phases. The outcome of each phase is a form which essentially consist of three sections: identification, status, and information. As with models of software development, SMM phases may overlap. Also it may be necessary to repeat one or more steps before a change is completed. Below is an overview of the individual phases:

- Change request: If the proposed change is corrective, a description of the error situation is included. For other types, a requirements specification is submitted.
- Change evaluation: Whereas a rejected proposed change is abandoned, a change approval form is created for an approved change. This together with the corresponding change proposal is the basic tool of the change management. By documenting new requirements, these forms become the contract between the requester of the change and the maintainers.

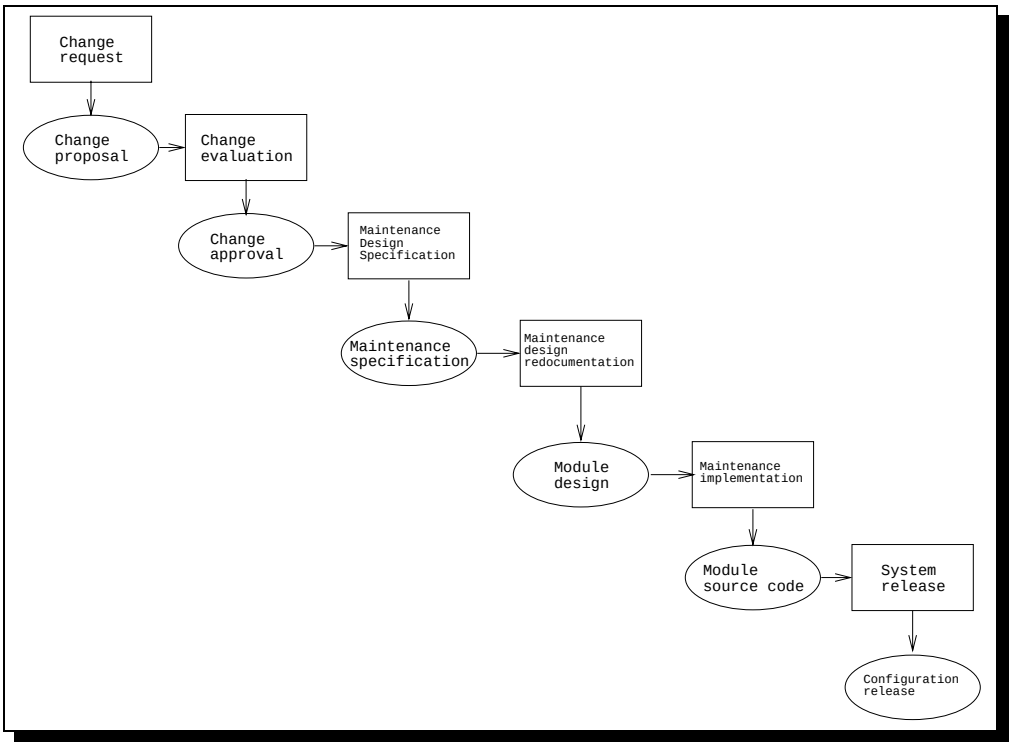

Figure 4.8: The software maintenance model

The approved changes are ranked and selected for the next release. Changes are batched by system releases. The work required is classified as perfective, adaptive, corrective, or preventive. The inadequacies described in the change proposal is identified in the application system. This involves different aspects for different kinds of changes.
  - Perfective: Identify new or altered requirements.
  - Adaptive: Identify the change in the environment.
  - Corrective: Identify repeatable error symptoms.
  - Preventive: Identify the deficiency in e.g. performance.
- Maintenance design phase: The result of this phase is the maintenance specification form. The design of a modification requires an examination of side-effects of changes. The maintainer must consider the software components affected and ensure that component properties are kept consistent. The integration and system test need to be planned and updated. Additionally, if the changes require new functionality, these are specified.
- Maintenance design re-documentation: This phase, along with the next, facilitates system comprehension by incremental re-documentation. The forms associated with these phases aim at documenting the software components of an existing application system.
- Maintenance implementation.
- System release: Validation of the overall system is achieved by performing integration and system test. The configuration release form contains details of the new application.

The specific functions of SCM as they influence the whole process is described below:

- Software configuration identification: The software configuration items in CONFORM is defined to be the SMM forms. SCM is able to control the release and changes in the forms throughout their existence, record and report their status, and verify their completeness and correctness.

- Software configuration control: In CONFORM this function is concerned with the control of changes made in an existing application system. Here, its role is to ensure that any change required in the application system is defined and implemented by following SMM phases. In so doing, change information is gradually recorded by the completion of the forms. This procedure ensures that all work performed to implement a change is traceable back to its change proposal, and that changes to an application system can only be made by authorized maintainers. Control of versions of SMM forms is also a task here.
- Software configuration status accounting: The implementation and effective use of this function in CONFORM is achieved by its automation and by the supporting information contained in the SMM forms.
- Software configuration auditing: The auditing function within CONFORM is the process which determines the overall acceptability of the proposed baseline at the end of each SMM phase. The process aims to establish the baselines of SMM forms. The auditing process in CONFORM can be divided into two separate phases. The first phase basically consist of performing checks in SMM forms for the completeness and correctness of the information. The outcome of the software tests which have been performed are also checked at this phase. The second phase aims at establishing all the baselines of the forms involved in a system release; thus the consistency and traceability between the SMM forms are checked.

Classification of CONFORM

- Weltanschauung: Objectivistic.
- Coverage in process: Maintenance.
- Coverage in product: A single application system.
- Reuse: Supported in the sense that maintenance is reuse by definition (See below).
- Conceptual modeling: Not mentioned.

4.2.9 Maintenance as Reuse-oriented Development

According to Basili all maintenance is in a sense reuse [21]. One can view a new version of an application system as a modification of the old or a new system that reuses much of the old system. Basili describes three maintenance models (Figure 4.9).

Quick fix methodology

The existing application system, usually just the source code, is taken as outset. The code and accompanying documentation is changed, and a re-compilation gives a new version. Figure 4.9a. demonstrates the change of the old system's source code to the new version's source code.

Iterative-enhancement methodology

Although this was originally proposed for development, it is well suited for maintenance. It assumes a complete and consistent set of documents describing the application system and

- starts with the existing requirements, design, code, and test documents.
- modifies the set of documents, starting with the highest level document affected by the changes, propagating the changes down through the full set of documents.

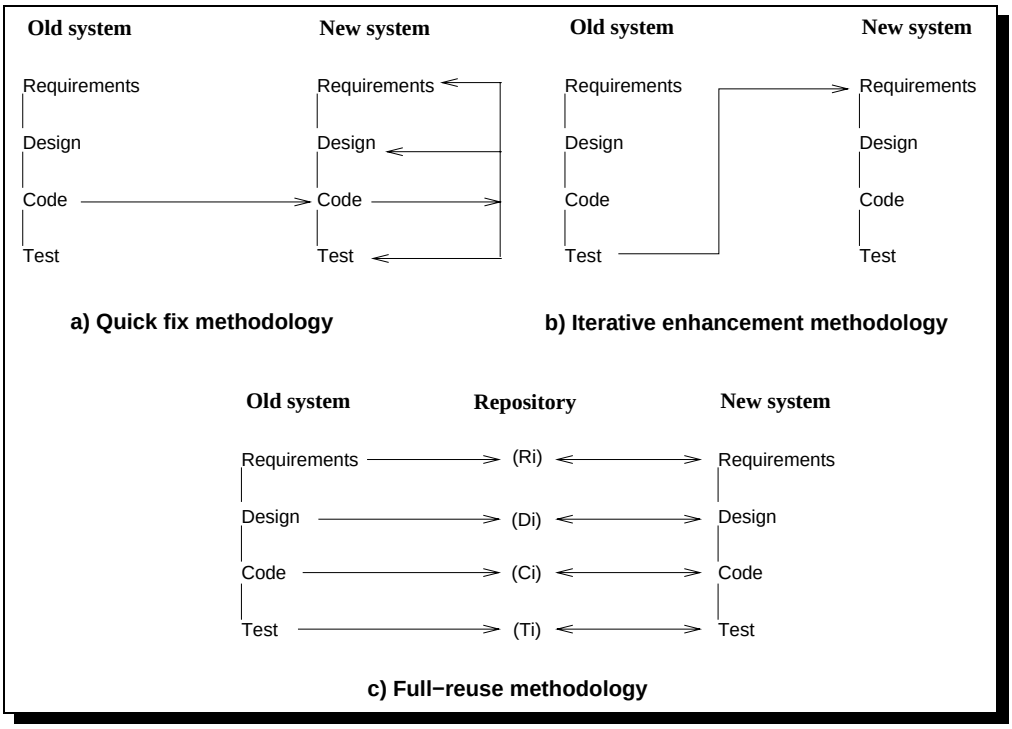

Figure 4.9: Maintenance process models

- at each step let you redesign the application system, based on the analysis of the existing system. An environment that supports the iterative-enhancement methodology also supports the quick-fix methodology.

Full-reuse methodology

A full reuse methodology starts with the requirements analysis and design of the new application system and reuses the appropriate requirements, design, and code from any earlier versions of the old system. It assumes a repository of artifacts defining earlier versions of the current application system and similar systems. An environment that support this also support the two other methodologies. According to Basili, one can consider development as a subset of maintenance. Maintenance environments differ from development environments in the constraints on the solution, customer demand, timeliness of response, and organization. Most maintenance organizations are set up for the quick-fix methodology, but not for the iterative enhancement or full reuse methodology, since they are responding to timeliness. This is best used when there is little chance the system will be modified again. This is the strength of the quick-fix methodology. Its weaknesses are that the modification is usually a patch that is not well-documented, the structure of the application system has partly been destroyed, making future evolution of the system difficult and error-ridden, and it is not compatible with development. The iterative-enhancement methodology allows redesign that let's the application system evolve, making future modifications easier. It is compatible with traditional development methodologies. It's a good methodology to use when the product will have a long life and evolve over time. In this case, if timeliness is also a constraint, one can use the quick-fix technique for patches and the iterative-enhancement methodology for long-term change. The drawbacks are that it is a more costly and possibly less timely technique

in the short run, and provides little support for generic artifacts or future similar systems.

The full-reuse methodology gives the maintainer the greatest degree of freedom for change, focusing on long range development for a set of application systems, which has the side effect of creating reusable artifacts of all kinds for future developments. It is compatible with development, and is according to Basili the way methodology should evolve. It is best used when you have multi-product environments or generic development where the portfolio has a long life. Its drawback is that it is more costly in the short run and is not appropriate for small modifications, although you can combine it with other models for such changes.

Classification of maintenance as reuse-oriented development

- Weltanschauung: Objectivistic.
- Coverage in process: All aspects of maintenance covered in an integrated manner (including development as a subset of maintenance). No particular mentioning of usage support.
- Coverage in product: Potential benefit for whole or large part of portfolio.
- Reuse: Development for and with reuse.
- Conceptual modeling: Not mentioned.

Whereas also the spiral model potentially covers maintenance, it do not distinguish between the need for sometimes doing quick-fixes. CONFORM is a combination of the quick-fix and iterative enhancement methodology.

4.2.10 Multiview

This methodology is based on several existing methodologies [17]. Multiview addresses the following areas:

- 1. How is the application system supposed to further the aims of the organization using it?
- 2. How can it be fitted into the working lives of the stakeholders in the organization?
- 3. How can the stakeholders best relate to the application system in terms of operating it and using the output from it?
- 4. What information processing functions are the application system to perform?
- 5. What is the technical specification of an application system that will come close enough to addressing the above questions?

The methodology is based on five main phases (Figure 4.10):

- 1. Analysis of human activity (answer to question 1).
- 2. Analysis of information (answer to question 4).
- 3. Analysis and design of socio-technical aspects (answer to question 2).
- 4. Design of the human-computer interface (answer to question 3).
- 5. Design of technical aspects (answer to question 5).

Not all development projects go through the same phases since the surroundings and particular circumstances differs from case to case. Multiview forms a flexible framework since different tools are available and are adjusted to different situations.

Analysis of human activity is based on the work of Checkland [51, 52] on Soft Systems Methodology (SSM). It focuses on finding the different stakeholders view of the world. Developers will with help from the users form a *rich picture* of the problem situation. Based on the rich picture the problems to be investigated in more detail may be extracted. The developers imagine and name systems that might help revealing the cause of the problem. Among suggested systems the developers and the users have to decide on a relevant system that is appropriate for the actual situation. Rich pictures are based on root definitions. A root definition is a concise verbal description

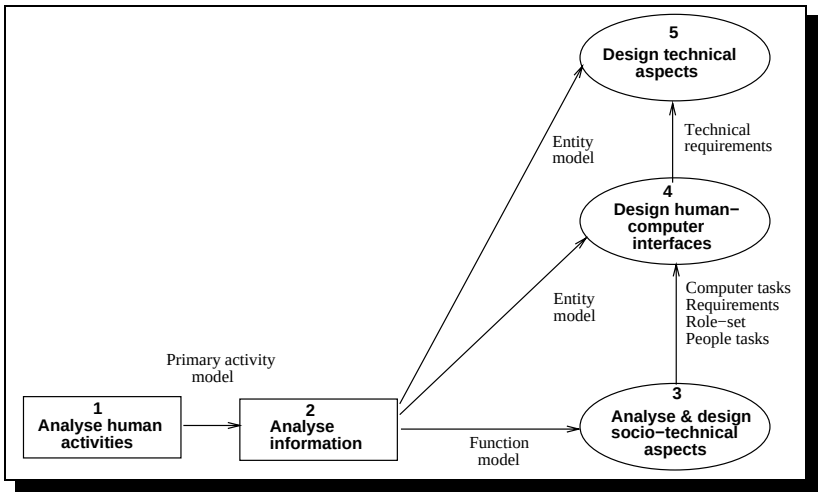

Figure 4.10: The phases and interdependencies in Multiview (From [17])

of the system which captures its essential nature. One technique to help come up with the root definitions is the 'CATWOE'-technique which answers the following question:

> **Who** is doing **what** for **whom**, and to whom are they **answerable**, what **assumptions** are being made, and in what **environment** is this happening?

Costumer is the 'whom', **A**ctor is 'who', **T**ransformation is 'what', **W**eltanschauung is 'assumptions', **O**wner is the 'answerable', and **E**nvironment is the environment.

An activity model show how the various activities are related to each other temporally. The activity model are a semi-formal conceptual model with some similarities with a DFD.

In information analysis, the three main tasks are development of a functional model, a data model, and interacting functions and entities and verifying the models.

Analysis and design of socio-technical aspects is concerned about the people using the IS. In order to develop a successful system, the system must be fitted into the working live of the users. The socio-technical methodology means that the technical and social aspects must fit each other in order to construct the best system. This phase is based on ETHICS [230].

Design of human-computer interfaces is concerned with the way users communicate with the CIS. Prototyping of user interfaces is supported.

In the design of technical aspects a technical solution is created in accordance with the requirements specified in early phases. What is considered is the entity model (phase 2), computer tasks (phase 3) and the human computer interfaces (phase 4).

Classification of Multiview

- Weltanschauung: Constructivistic. This is based on the application of SSM.
- Coverage in process: Primarily focused on development, but have special emphasis on preparing the usage-situation, even if actual usage support is not covered.
- Coverage in product: A single application system.
- Reuse: Not explicitly supported.
- Conceptual modeling: Used actively. The languages used for modeling are semi-formal or informal.

4.2.11 STEPS

STEPS (Software Technology for Evolutionary Participative System Development) was originally developed at the University of Berlin [90]. The development is based on work by Mumford [230] and on work within the so-called "Scandinavian school" [80].

The methodology was developed to fill the gap between the user-aspects and the technical aspects. The main idea is that application development should be suited to the work-processes/users. The development does not start because of defined problems as in traditional system development, but as a learning process about the users' situation. In a project there will always exist different perspectives among users and management and among users themselves. This methodology states that applying a multi-perspective is a prerequisite for cooperative work. The cooperation should enhance user competence and is also a learning process for the developers. Sometimes users and developers are meant to work together and other times they work separately on different tasks as illustrated in Figure 4.11.

The methodology is evolutionary with cycles resulting in new versions of application systems. This support both incremental development and traditional maintenance projects. Prototyping is used for exploratory and experimental purposes during user-interface design. Prototyping also enables users and developers to discuss different suggestions to solutions.

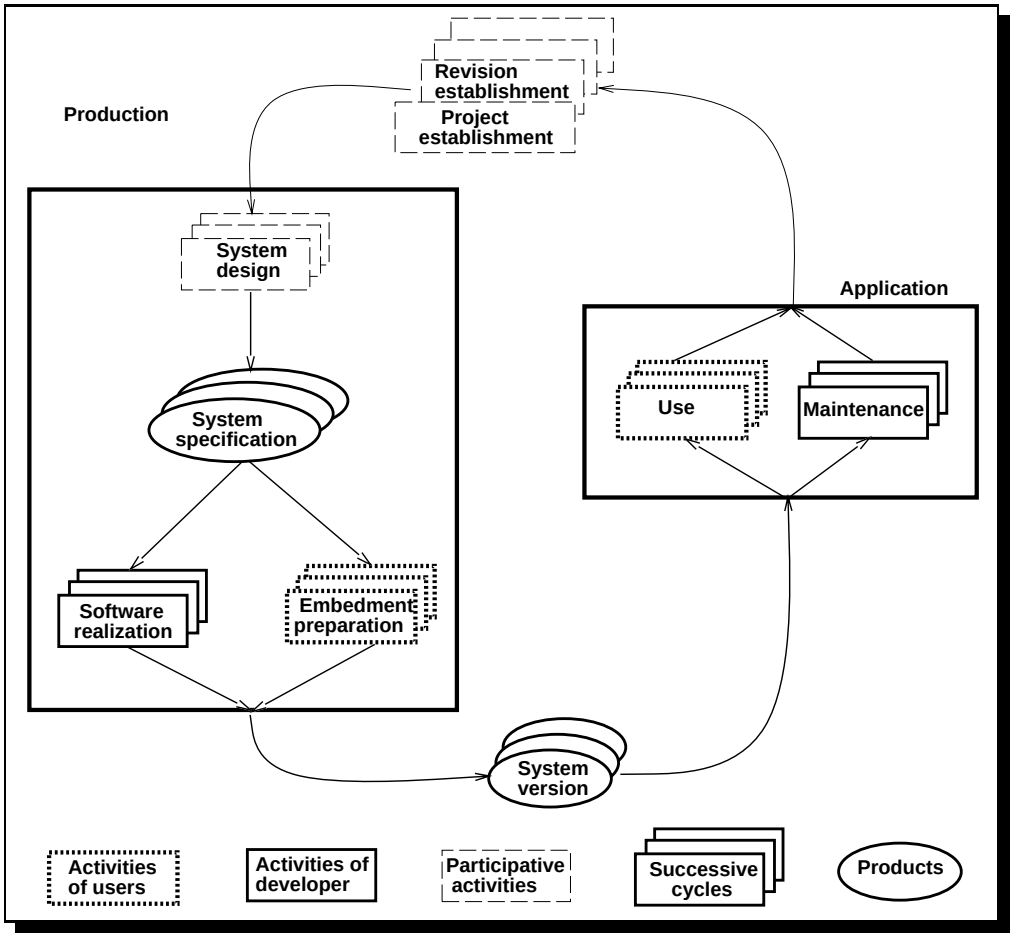

Figure 4.11: The STEPS methodology (From [90])

The project is established by the users and the developers in cooperation. Subjects to be agreed

upon in this phase are the project goals, how the goals are going to be reached, system concepts and system strategy. During the establishment of a revision project, the system concept and the project plan are updated. System design is the next phase. Both developers and users are supposed to take part and they must agree upon issues like working activities, work organization, requirements to system functions, data structures, user interface etc. This is performed in cycles of analysis, synthesis, and revision and it is mainly focused on communication between developers and users and the mutual learning process as the creative part of work leading to insights.

Formalization, description, and implementation based on the specification are tasks which only include the systems developers. If changes in the implementation have to be made, the methodology says nothing about possible cooperation between the developers and users in this phase. Cooperation between user participants and developers is not considered in the use and maintenance phase.

The users who participate in the system design is not supposed to need any previous knowledge about system design or methods. All relevant knowledge will be acquired during the design process. This probably lead to higher costs in the first phases of system development, but the extra costs are meant to be payed off in terms of user quality and acceptance later on.

The project model differs from the conventional model in several ways [90]:

- It is cyclic rather than linear, portraying software development as proceeding in system versions, all development steps and defining documents being subject to revision.
- It combines software production and application, visualizing the tasks of both developers and users. The development of each application system is fitted to an associated reorganization of user work processes; the interplay between these two is anticipated in the cooperative design and evaluated in the cooperative revision step.
- It refers to a class of potential development strategies allowing the choice of a situation specific strategy as needed in the project at hand rather than depicting one ideal development strategy to be copied as closely as possible in all projects
- It relies minimally on pre-defined intermediate artifacts.
- It is not defined in terms of the domains of discourse relevant in software development.

Classification of STEPS

- Weltanschauung: Constructivistic.
- Coverage in process: Covers development, use, and maintenance in an integrated manner. Do not discuss emergency error-correction.
- Coverage in product: A single application system.
- Reuse: Not supported except in the sense of maintenance as reuse in the large.
- Conceptual modeling: Not explicitly mentioned, although incorporation of support of object-orientation has been planned.

4.3 Chapter Summary

We have in this chapter presented some methodologies and methodological frameworks for support of COIS. A short summary of our classifications is given in Table 4.2. According to this we conclude the following:

- Weltansscahuung: As also noted by Hirschheim and Klein [134], most earlier and current methodologies for application systems development and maintenance have an objectivistic

Methodology	Weltanschauung	Process coverage	Product coverage	Reuse	Conceptual modeling
Waterfall	Objectivistic	Development +	one	-	Little
Structured	Objectivistic	Development	one	-	Semi-formal
Prototyping	Objectivistic/ constructivistic	Development (early)	one	with	Potentially
Operational	Objectivistic	Development	one	Generative	Formal
Method/1	Objectivistic	Development +	one/portfolio	In the large	Semi-formal
Spiral	Objectivistic	Development/ maintenance	one	with	-
Hierarchical spiral	Objectivistic	Development/ maintenance	one	-	Yes
Fountain	Objectivistic	Development (mainly)	one	for/with	OOA
REBOOT	Objectivistic	Development/ maintenance	one/portfolio	for/with	OOA
CONFORM	Objectivistic	Maintenance	one	-	-
Basili	Objectivistic	Development/ maintenance	one/portfolio	for/with	-
Multiview	Constructivistic	Development	one	-	Semi-formal
STEPS	Constructivistic	Development maintenance/use	one	-	-

Table 4.2: Classification of Methodologies

outlook. Some exceptions illustrated in this chapter are STEPS [90] and Multiview [17]. Other examples are methodologies based on SSM [51] and some PD-methodologies [276].

- Coverage in process: Most methodologies for CIS-support are focused on development, with maintenance being looked upon as a separate end-phase. Several methodologies focused on maintenance also exist (e.g.CONFORM [43], see also [30]), even if this part of CIS-support is not shown the same interest as development by researchers according to [121, 158]. Some methodologies covers both development and maintenance in the same framework in an integrated manner (e.g. The Spiral Model [29], The Hierarchical Spiral Model [140, 139] and the framework presented by Basili [21] where also emergency error-correction is covered). STEPS [90] also includes the usage aspect.
- Coverage in product: We have found few methodologies that cover traditional development or maintenance of the whole portfolio in a focused manner, even though maintenance can be said to often be performed in this way [299]. Several methodologies include organization-wide CIS-planning (e.g. METHOD/1 [8]). Approaches for enterprise-modeling (e.g. [275]) are also in this category.
- Reuse: Some methodologies explicitly addressing reuse exist (e.g. REBOOT [161]), even if few development and maintenance methodologies are geared towards conscious component reuse. Operational and transformational approaches as described in [345] are highly geared towards generative reuse.

- Use of conceptual models: Many methods use conceptual modeling to some extent, even if most use only semi-formal modeling languages. On the other hand, the use of operational conceptual models have received increased interest and will be further discussed in Chapter 5 and Chapter 7.

None of the presented methodologies can be said to cover all aspects of the classification in a totally satisfactory way. All this said, there seems to be an overall view that there are no right methodology for all situation [17, 90, 105, 140] something which are also recognized in more traditional methodologies like Method/1. The different development and maintenance efforts can vary according to several factors e.g.:

- The wickedness of the problem (cf. Rittel in Chapter 2.2).
- The complexity of the application system (cf. Brooks in Chapter 2.2).
- The current state of flux (cf. Chapter 1.2).
- The size, perceived importance, and risks of performing the changes (cf. Boehm in Section 4.2.4.)
- The number of stakeholders affected, skills needed, and possessed (cf. Chapter 3).
- The number of different views of the situation (cf. social construction theory, Chapter 1.1).

Thus there is a need for flexibility, but in our opinion one still need a methodology or a methodological framework of some sort to be able to deliver CIS-support in an organization. Taking into account the multitude of techniques, there is an obvious need for an integrative framework that can incorporate existing more detailed approaches and support their flexible situation-dependant use.

Chapter 5

Conceptual Modeling

In this chapter we will discuss the use of conceptual modeling in the light of social construction theory, and give an overview of different perspectives to conceptual modeling.

When discussing conceptual modeling below, we apply the following limitations:

- The languages for conceptual modeling that we are regarding are mainly diagrammatic, and have a limited vocabulary. The main symbols of the languages depicts what in many connections are referred to as *concepts* e.g. states, processes, entities, and objects. We have generally used the terms 'phenomena' and 'phenomena classes' instead of 'concepts' in this thesis since this word is used in many different meanings in natural language.
- The use of the conceptual models that we are regarding is primarily as an intermediate representation used in the development and maintenance of CISs. We recognize that conceptual modeling languages can be useful also for other purposes e.g. organizational modeling with no immediate implementation in mind, but will not treat other uses in detail in this thesis as long as they are not part of CIS-support on some level.
- The described languages are meant to have a general applicability, i.e. they are not made specifically for the modeling of a limited area.

We combine in this thesis the use of conceptual models with a philosophical outlook of social construction of reality. Some researchers might react negatively to this, taking their perception of what conceptual modeling is. Klein and Lyytinen [164] state that most of the current modeling approaches for data and rule modeling are based on an objectivistic ontology. This is exemplified by quotes such as that "the real world consists of entities and relationships" [53]. However, as noted also by Lyytinen and Klein, these assumptions have not been universally shared. In [290] for instance, it appears that what is modeled is some users *perception* of the "real world".

As observed by Lewis [191], the present practice of data modeling includes a large element of subjectivity. This subjectivity exists whether or not the data-focused approaches uses 'entity', 'object', or 'phenomenon' as ontological devices. If entities *are* taken to have real world existence, then the participants in the modeling effort must be choosing from the infinitely large number of entities that exist, only some entities being relevant and suitable for inclusion in the model. Consequently, the process of creating such a model is not value-free and the resulting conceptual model is not unbiased. If real world existence is *not* assumed, then the conceptual models are epistemological devices and entities or objects are, by definition, subjectively created by the participants in the modeling effort to understand their perception of the world. In either case, the conceptual model can only be described as an interpretation of "reality".

Thus, in both cases will it be useful to admit to this subjectivity and allow several models to co-exist even if only one of them will be externalized in the organization in the form of an appli-

cation system within a time interval. Examples of existing approaches explicitly acknowledging several realities are some approaches grown out from object-orientation [127, 267]. We also saw in Chapter 4 that Multiview [17] having a constructivistic world-view use traditional conceptual languages as an important part of the methodology. A similar attempt to integrate SSM [51] and software engineering approaches is reported in [77]. On the other hand, even if traditional conceptual modeling languages can be used having a constructivistic world-view, they do not have explicit constructs for capturing differing views directly in the model and making this visible to the audience of the model. They neither have the possibility to differentiate between the rules of necessity and the deontic rules represented.

In this connection it is also important to be aware of that CISs themselves are models of someones perceived reality, even if these models are often not so obvious due to the amount of details present and the way they are obscured by the programming languages used [31].

Conceptual modeling languages are biased towards a special way of perceiving the world. This has both positive and negative effects:

- The languages have constructs which force the analyst and users to perceive the world in some specific way emphasizing some aspects, whereas neglecting others.
- The more the analysts and users work with one specific language, the more their thinking will be influenced by this, and their awareness of the aspects of their perception of the world that do not fit in might be diminished cf. the Sapir-Whorf hypothesis [295].
- For those kinds of problems for which the approach is suitable, the neglect of features that are not covered can have a positive effect, in that one more easily manages to concentrate on the relevant and important issues. However, it is hard to know what issues are relevant and different issues and ways of describing the world can be relevant for different people at the same time.

5.1 Conceptual Modeling as Social Construction

Development and maintenance of conceptual models in different languages follows a similar pattern to what is described in Chapter 1.1.

The construction of a conceptual model of "reality" as it is perceived by someone, is partly a process of externalization of parts of this persons internal reality, and will in the first place act as organizational reality for the additional audience of the model. This model can then be used in the sense-making process by the other stakeholders, internalizing the views of the other if they are found appropriate. This internalization is based on pre-understanding, which includes assumptions implicit in the languages used for modeling. The language in turn is learned through internalization.

After reaching a sufficiently stable shared model one might wish to externalize this in a more material way, transferring it to the organization in the form of computer technology. Here a new need for internalization of the technology is needed for the CIS to be useful for the part of the organization that is influenced by it. Also here, it should be possible to utilize the conceptual models to understand what the CIS does, and most importantly, *why* it does it. Making sense of the technology is also important to be able to change it, and the already developed conceptual models can act as a starting point for additional maintenance efforts on the CIS when deemed necessary.

It should be noted that the ability and possibility for the different social actors of the organization to externalized their local reality will differ in several ways. Since the languages, and types

of languages used are usually already defined when a decision to create an application system has been made, persons with a long experience in using this kind of languages will have an advantage in the modeling process. This applies especially to the specialist on computer technology (e.g analysts). This is not necessarily bad, if they did not have this knowledge it would not be interesting to include them in the development process in the first place. Rather it is important to be aware of this difference, to avoid the most apparent dangers of model monopoly as discussed by Bråten [34]. What is also apparent is that also some persons in the organization have a greater possibility to externalize their reality than others, both generally e.g. the financiers of an endeavour will usually, implicitly or explicitly, be in a position that will bias a solution in their perceived favor, and specifically by the use of certain modeling techniques. Gjersvik has illustrated how the way management perceive the world can be more easily externalized in a CIS than the way shop-floor workers perceive the world [102]. We will return to how these problems can be attacked in Chapter 8 and Chapter 10.

The use of conceptual models being constructed as part of the development and maintenance of application systems has been discussed by several researchers [37, 73, 147, 182, 327]. This discussions can be summarized as follows:

- Representation of systems and requirements: The conceptual model represents properties of the problem area and perceived requirements to the information system. A conceptual model can give insight into the problems motivating the development project, and can help the systems developers and users understanding the application system to be built. Moreover, by analyzing the model instead of the business area itself, one might deduce properties that are difficult if not impossible to perceive directly since it is possible to concentrate on only some aspects at the time.
- Vehicle for communication: The conceptual model can serve as a means for sense-making and communication among stakeholders. By hopefully bridging the realm of the end-users and the CIS, it opens for a more reliable and constructive exchange of opinions between users and the developers of the CIS, and between different users. The models both help and restrict the communication by establishing a nomenclature and a definition of phenomena in the modeling domain.
- Basis for design and implementation: The conceptual model can act as a prescriptive model, to be approved by the users and other stakeholders who specify the desired properties of a CIS. The model can establish the content and boundary of the area under concern more precisely. When designing and implementing the CIS, the relevant parts of the model are guiding the development process. Similarly, the design and implementation might afterwards be tested against the model to make sure that the different representations are consistent. When the model is formal and contains sufficient detail, it is often possible to produce the application system more or less directly from the model.
- Documentation and sensemaking: The conceptual model is an easily accessible documentation of the CISs which are applied in the organization. Due to its independence of the implementation, it is less detailed than other representations while still representing the basic functionality of the system. Compared to manually produced textual documentations, the conceptual model is easier to maintain since it is constructed as part of the process of developing and maintaining the application system in the first place. With the introduction of more flexible methodologies and tool support, conceptual models are also likely to be used in reverse and re-engineering, and when reusing artifacts constructed in connection with other application systems.

Summing up, a conceptual model is used both for communication and representation, and faces

demands from both social and technical actors. As a consequence of this duality, requirements for conceptual modeling languages will pull in opposite directions. We will investigate this in more detail in Chapter 6 after introducing examples of conceptual modeling languages covering different modeling perspectives.

5.2 Overview of Languages for Conceptual Modeling

In this section, we survey "the state of the art" of modeling languages, including those that have been applied in mature methodologies for system development and maintenance and some that are still on the research level. The overview will concentrate on the basic components and features of the languages to illustrate different ways of abstracting human perception of reality.

Modeling languages can be divided into classes according to the core phenomena classes that is represented in the language. We have called this the main *perspective* of the language. A traditionally distinction is between the structural, functional, and behavioral perspective [242]. Yang [340] identifies a full perspective to include the following based on [195, 323]:

- Data perspective. This is parallel to the structural perspective.
- Process perspectives. This is parallel to a functional perspective.
- Event/behavior perspective. The conditions by which the processes are invoked or triggered. This is covered by the behavioral perspective.
- Role perspectives. The roles of various actors carrying out the processes of a system.

In F3 [40], it is recognized that a requirement specification should answer the following questions:

- Why is the system built?
- Which are the processes to be supported by the system?
- Which are the actors of the organization performing the processes?
- What data or material are they processing or talking about?
- Which initial objectives and requirements can be stated regarding the system to be developed.?

This indicate a need to support what we will term the *rule-perspective*, in addition to the other perspectives mentioned.

Finally, in the NATURE project [147], one distinguishes between four worlds: Usage, subject, system, and development. Conceptual modeling as we use it applies to the subject and usage world for which NATURE propose data models, functional models, and behavior models, and organization models, business models, speech act models, and actor models respectively.

In addition can modeling languages be classified according to their time-perspective [293]:

- Static perspective: Provide facilities for describing a snapshot of the perceived reality, thus only considering one state.
- Dynamic perspective: Provide facilities for modeling state transitions, considering two states, and how the transition between the states take place.
- Temporal perspective: Allow the specification of time dependant constraints. In general, sequences of states are explicitly considered.
- Full-time perspective: Emphasize the important role and particular treatment of time in modeling. The number of states explicitly considered at a time is infinite.

Based on the above, to give a broad overview of the different perspectives state-of-the-art conceptual modeling approaches accommodate, we have focused on the following perspectives:

- Structural perspective
- Functional perspective

- Behavioral perspective
- Rule perspective
- Object perspective
- Communication perspective
- Actor and role perspective

That a language is classified as having e.g. a functional perspective do not mean that it can not represent e.g. structural phenomena, but rather that it emphasizes the representation of processes. Many of the languages presented here are often used together with other languages in so-called combined approaches. More detailed examples of combined approaches will be given in Chapter 7.

5.2.1 The Structural Perspective

The structural perspective have traditionally been handled by languages for data modeling. Whereas the first semantic data modeling language was published in 1974 [138], the first having major impact was the the *entity-relationship* language of Chen [53].

Basic vocabulary and grammar of the ER-language: In [53], the basic components are:

- **Entities**. An *entity* is a phenomenon that can be distinctly identified. Entities can be classified into entity classes;
- **Relationships**. A *relationship* is an association among entities. Relationships can be classified into relationship classes;
- **Attributes and data values**. A value is used to give value to a property of an entity or relationship. Values are grouped into value classes by their types. An *attribute* is a function which maps from an entity class or relationship class to a value class; thus the property of an entity or a relationship can be expressed by an attribute-value pair.

An ER-model contains a set of entity classes, relationship classes, and attributes. An example illustrating the diagrammatic representation is given in Figure 5.1, the rectangles being entity classes, the diamond a relationship class, and the ovals attribute classes. The figure is part of a data model for a conference example. This example will be used also later on in the thesis.

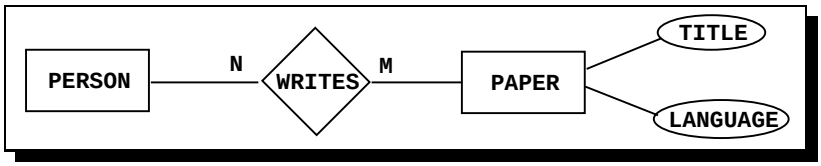

Figure 5.1: Example of an ER-model

Several improvements have later been proposed for so-called semantic data modeling languages [138, 253].

Basic vocabulary and grammar for semantic data modeling language: In Hull and King's overview [138] a generic semantic modeling language (GSM) is presented. Figure 5.2 illustrates the vocabulary of GSM:

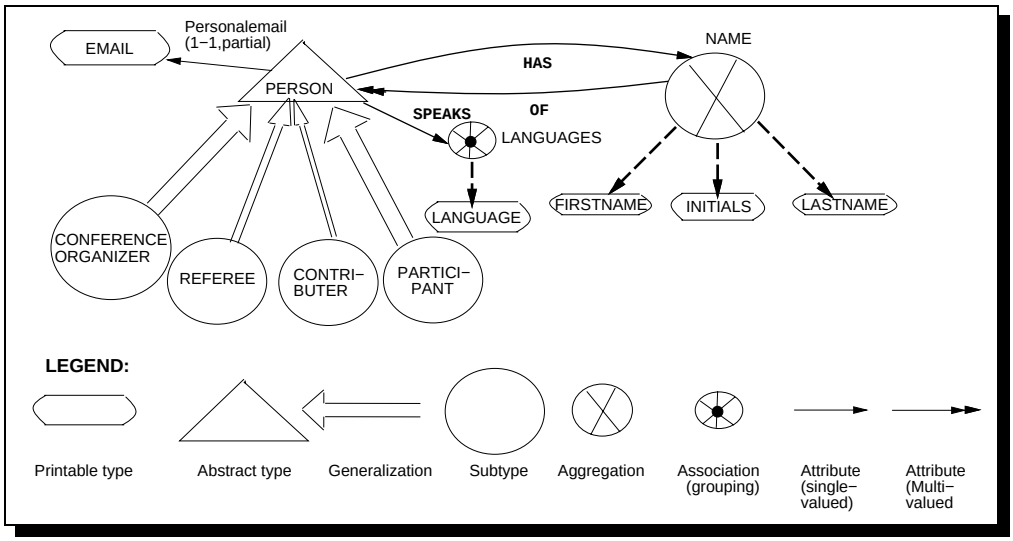

Figure 5.2: Example of a GSM model

- **Primitive types**. The data types in GSM are classified into two kinds: the printable data types, that are used to specify some visible values, and the abstract types that represent some entities.
- **Constructed types built by means of abstraction**. The most often used constructors for building abstractions are generalization, aggregation, and association.
- **Attributes**.

In addition it is possible to specify derived classes in GSM.

Relationships between instances of types may be defined in different ways. We see in Figure 5.2 that a relationship is defined by a two-way attribute(an attribute and its inverse). In the ER modeling language, a relationship is represented as an explicit type. The definition of relationship types provides the possibility of specifying such relationships among the instances of more than two types as well as that of defining attributes of such relationship types.

Conceptual graphs: Conceptual graphs [294] is a language based on linguistics, psychology, and philosophy. In the models, concept nodes represent entities, attributes, states, and events, and relation nodes show how the concepts are interconnected. A conceptual graph is a finite, connected, bipartite graph. Every conceptual relation has one or more arcs.

Each conceptual graph asserts a single proposition and has no meaning in isolation. Only through a semantic network are its concepts and relations linked to context, language, emotion, and perception. Figure 5.3 shows a conceptual graph for *a cat sitting on a mat.* Dotted lines link the nodes of the graph to other parts of the semantic network.

- Concrete concepts are associated with percepts for experiencing the world and motor mechanisms for acting upon it.
- Some concepts are associated with the vocabulary and grammar rules of a language.
- A hierarchy of concept types defines generalization relationships between concepts.
- Formation rules determine how each type of concept may be linked to conceptual relations.
- Each conceptual graph is linked to some context or episode to which it is relevant.
- Each episode may also have emotional associations, which indirectly confer emotional overtones on the types of concepts involved.

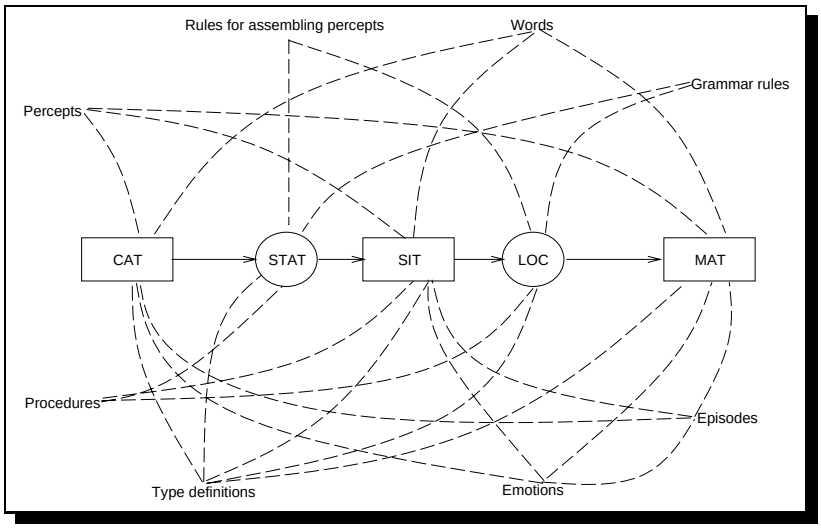

Figure 5.3: A conceptual graph linked to a semantic network (From [294])

5.2.2 The Functional Perspective

The main phenomena class in the functional perspective is the process: A process is defined as an activity which based on a set of *phenomena* transforms them to a possibly empty set of *phenomena.*

The best know conceptual modeling language with a process perspective is data flow diagrams (DFD) [97] which describes a situation using the symbols illustrated in Figure 5.4:

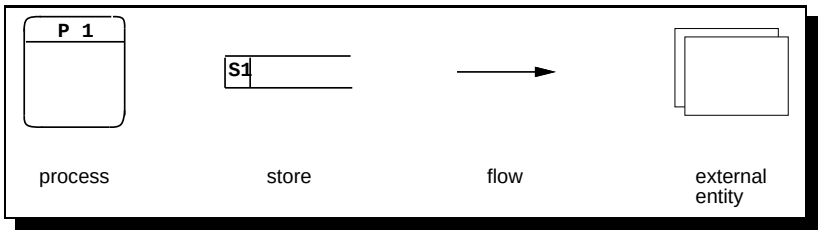

Figure 5.4: Symbols in the DFD language

- **Process**. Illustrates a part of a system that transforms a set of inputs to a set of output.
- **Store**. A collection of data or material.
- **Flow**. A movement of data or material within the system, from one system component (process, store, or external entity) to another;
- **External entity**. An individual or organizational actor, or a technical actor that is outside the boundaries of the system to be modeled, which interact with the system.

With these symbols, a system can be represented as a network of processes, stores and external entities linked by flows. A process can be decomposed into a new DFD. When the description of the process is considered to have reached a detailed level where no further decomposition is needed, “process logic” can be defined in forms of e.g. structured English, decision tables, and decision trees.

When a process is decomposed into a set of sub-processes, the sub-processes are grouped around the higher level process, and are co-operating to fulfill the higher-level function. This view on DFDs has resulted in the "context diagram" [97] that regards the whole system as a process which receives and sends all inputs and outputs to and from the system. A context diagram determine the boundary of a system. Every activity of the system is seen as the result of a stimulus by the arrival of a data flow across some boundary. If no external data flow arrives, then the system will remain in a stable state. Therefore, a DFD is basically able to model reactive systems.

DFD is a semi-formal language. Some of the short-comings of DFD regarding formality are addressed in the transformation schema presented by Ward [325]. The main symbols of his language are illustrated in Figure 5.5.

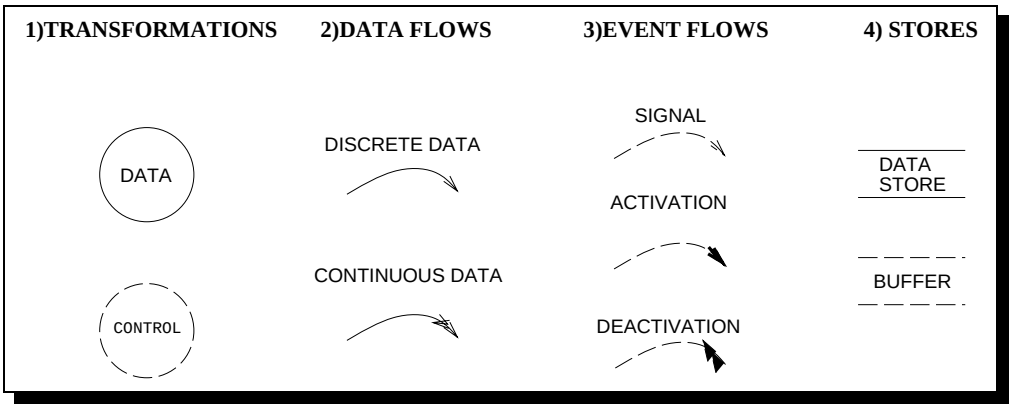

Figure 5.5: Symbols in the transformation schema language

There are four main classes of symbols:

- **1. Transformations**: A solid circle represent a data transformation, which are used approximately as a process in DFD. A dotted circle represents a control transformation which controls the behavior of data transformations by activating or deactivating them, thus being an abstraction on some portion of the systems' control logic.
- **2. Data flows**: A discrete data flow is associated with a set of variable values that is defined at discrete points in time. Continuous data flows are associated with a value or a set of values defined continuously over a time-interval.
- **3. Event flows**: These report a happening or give a command at a discrete point in time. A signal shows the sender's intention to report that something has happened, and the absence of any knowledge on the sender's part of the use to which the signal is put. Activations show the senders intention to cause a receiver to produce some output. A deactivation show the senders intention to prevent a receiver from producing some output.
- **4. Stores**: A store acts as a repository for data that is subject to a a storage delay. A buffer is a special kind of store in which flows produced by one or more transformations are subject to a delay before being consumed by one or more transformations. It is an abstraction of a stack or a queue.

Both process and flow decomposition are supported.

Whereas Ward had a goal of formalizing DFD's , Opdahl and Sindre [245, 247] try to adapt data flow diagrams to what they term 'real-world modeling'.

Problems they note with DFD in this respect are as follows:

Phenomena class	Process	Flow	Store
Activity	Transformation	Transportation	Preservation
Aspect	Matter	Location	Time

Table 5.1: A data flow diagram taxonomy of real-world dynamics

- 'Flows' are semantically overloaded: Sometimes a flow means transportation, other times it merely connects the output of one process to the input of the next.
- Parallelism often has to be modeled by duplicating data on several flows. This is all right for data, but material cannot be duplicated in the same way.
- Whereas processes can be decomposed to contain flows and stores in addition to sub-processes, decomposition of flows and stores is not allowed. This makes it hard to deal sensibly with flows at high levels of abstraction [39].

These problems have been addressed by unifying the traditional DFD vocabulary with a taxonomy of real-world activity, shown in Table 5.1: The three DFD phenomena "process," "flow", and "store" correspond to the physical activities of "transformation," "transportation", and "preservation" respectively. Furthermore, these three activities correspond to the three fundamental aspects of our perception of the physical world: matter, location, and time. Hence, e.g., an *ideal flow* changes the location of items in zero time and without modifying them.

Since these ideal phenomena classes are too restricted for high level modeling, real phenomena classes were introduced. Real processes, flows, and stores are actually one and the same, since they all can change all three physical aspects, i.e., these are fully inter-decomposable. The difference is only subjective, i.e., a real-world process is mainly perceived as a transformation activity, although it may also use time and move the items being processed. Accordingly, real flows and stores are mainly perceived to be transportation and preservation activities, respectively, although both may also transform the items handled.

Additionally, the problem with the overloading of 'flow' is addressed by introducing a *link*, for cases where there is no transportation. Links go between *ports* located on various processes, stores and flows, and may be associated with spatial coordinates

[245] also provides some definitions relating to the *items* to be processed, including proper distinctions between data and material. Items have *attributes* which represent the properties of data and materials, and they belong to *item classes.* Furthermore classes are related by the conventional abstraction relations aggregation, generalization, and association. Hence the specification of item classes constitute a *static* real-world model which complements the dynamic real-world models comprising processes, flows, stores, and links.

The symbols in the language are shown in Figure 5.6. The traditional DFD notation for processes and flows are retained, however, to facilitate the visualization of decomposition, it is also possible to depict the flow as an enlarged kind of box-arrow. Similarly, to facilitate the illustration of decomposed stores, full rectangles instead of open-ended ones are used. Links are shown as dotted arrows.

5.2.3 The Behavioral Perspective

In most languages with a behavioral perspective the main phenomena are states and transitions between states. State transitions are triggered by events [60].

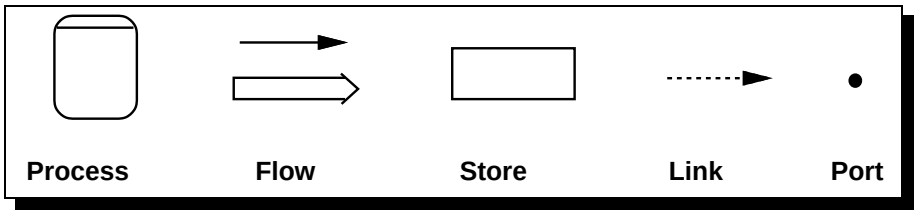

Figure 5.6: Symbols in the real-world modeling language

The vocabulary of state transition diagrams is illustrated in Figure 5.7 and are described below:

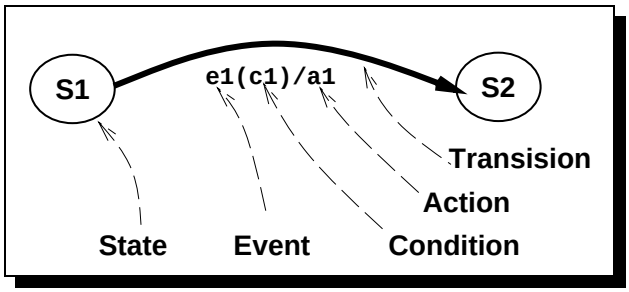

Figure 5.7: Symbols in the state transition modeling language

- **State**: A system is always in one of the states in the lawful state space for the system. A state is defined by the set of transitions leading to that state, the set of transitions leading out of that state and the set of values assigned to attributes of the system while the system resides in that state.
- **Event**: An event is a message from the environment or from system itself to the system. The system can react to a set of predefined events.
- **Condition**: A condition for reacting to an event. Another term for this is 'guard'.
- **Action**: The system can perform an action in response to an event.
- **Transition**: Receiving an event will cause a transition to a new state if the event is defined for the current state, and if the condition assigned to the event evaluates to true.

A simple example that indicates the state of a paper during the preparation of a professional conference is depicted in Figure 5.8.

Abstraction mechanisms are added to traditional STD in Statecharts [124] to provide the language with modularity and hierarchical construct as illustrated in Figure 5.9.

- **XOR decomposition**: A state is decomposed into several states. An event entering this state will have to enter one and only one of its sub-states. Thus generalization is supported.
- **AND decomposition**: A state is divided into several states. The system resides in *all* these states when entering the decomposed state. Thus aggregation is supported.

One has introduced the following mechanisms to be used with these abstractions:

- **History**: When entering the history of a XOR decomposed state, the sub-state which was visited last will be chosen.
- **Deep History**: The semantics of history repeated all the way down the hierarchy of XOR decomposed states.

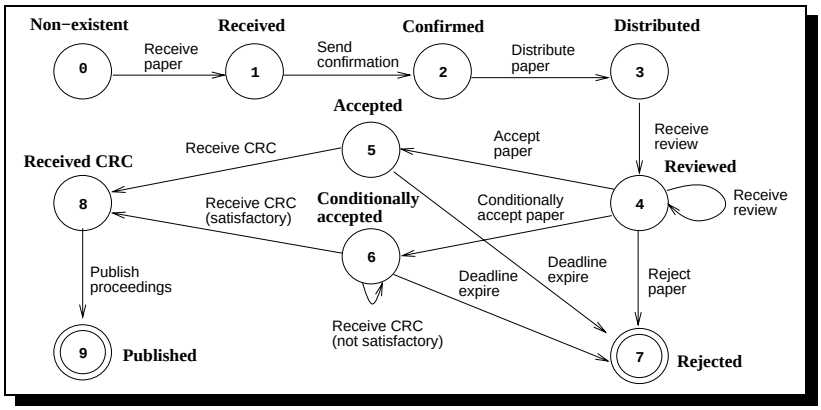

Figure 5.8: Example of a state transition model

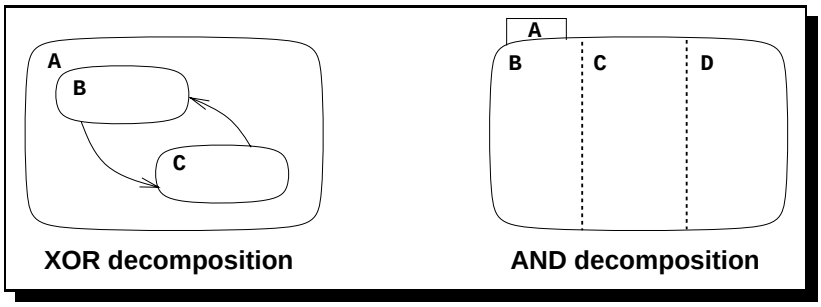

Figure 5.9: Decomposition mechanisms in Statecharts

- **Condition**: When entering a condition inside a XOR decomposed state, one of the sub-state will be chosen to be activated depending on the value of the condition.
- **Selection**: When entering a selection in a state, the sub-state selected by the user will be activated.

In addition support for the modeling of delays and time-outs is included.

Petri-nets

Petri-nets [258] is another well-known behaviorally oriented modeling language. A model in the original Petri-net language is shown in Figure 5.10. Here, places indicate a system state space, and a combination of tokens included in the places determine the specific system state. State transitions are regulated by firing rules: A transition is enabled if each of its input places contains a token. A transition can fire at any time after it is enabled. After the firing of a transition, a token is removed from each of its input places and a token is produced in all output places.

Figure 5.10 shows how dynamic properties like precedence, concurrency, synchronization, exclusiveness, and iteration can be modeled in a Petri-net. The associated model patterns along with the firing rule above establish the execution semantics of a Petri-net.

The classical Petri net cannot be decomposed. This is inevitable by the fact that transitions are instantaneous, which makes it impossible to compose more complex networks (whose execution is bound to take time) into higher level transitions. However, there exists several more recent dialects of the Petri net language (for instance [213])) where the transitions are allowed to take

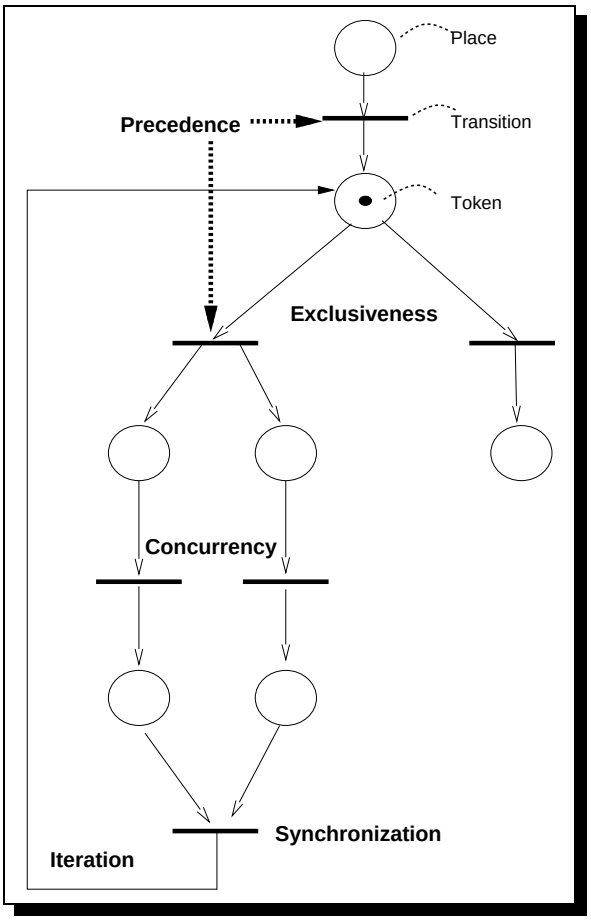

Figure 5.10: Dynamic expressiveness of Petri-nets

time, and these approaches provide decomposition in a way not very different from that of a data flow diagram. Timed Petri Nets [213] also provide probability distributions that can be assigned to the time consumption of each transition and is particularly suited to performance modeling. BNM [292] introduces extended firing rules using pre and post-conditions to further define the transition criteria. Moreover, the dynamic part is linked to a structural part, thus having a so-called colored Petri-net (i.e. the tokens are typed).

5.2.4 The Rule Perspective

A rule has been defined as follows:

> A **rule** is something which influences the *actions* of a non-empty set of *actors.* A rule is either a rule of necessity or a deontic rule [327].
>
> A **rule of necessity** is a *rule* that must always be satisfied. It is either analytic or empirical (see below).
>
> A *rule of necessity* which can not be broken by an inter-subjectively agreed definition of the terms used in the rules is called **analytic**.
>
> A *rule of necessity* that can not be broken according to present *shared explicit knowledge* is called **empirical**.

A **deontic rule** is a *rule* which is only socially agreed among a set of persons. A deontic rule can thus be violated without redefining the terms in the rule. A deontic rule can be classified as being an obligation, a recommendation, a permission, a discouragement, or a prohibition [180].

A **constitutive rule** is a *deontic rule* which applies to *phenomena* that exist only because the rule exist.

The general structure of a rule is

" if *condition* then *expression*"

where *condition* is descriptive, indicating the scope of the rule by designating the conditions in which the rule apply, and the *expression* is prescriptive. According to [312] any rule, however expressed, can be analyzed and restated as a compound conditional statement of this form.

Current applications Representing knowledge by means of rules is not a novel idea. According to Davis and King [64], production systems were first proposed as a general computational mechanism by Post in 1943. Today, rules are used for knowledge representation in a wide variety of applications, such as expert systems, tutoring and planning systems, database systems, and finally requirement specification. It is the use of rules within requirement specification that will be our focus here.

Several advantages have been experienced with a declarative, rule-based approach to information systems modeling:

- *Problem-orientation:* The representation of business rules declaratively is independent of what they are used for and how they will be implemented. With an explicit specification of assumptions, rules and constraints, the analyst has freedom from technical considerations to reason about application problems [66, 117]. This freedom is even more important for the communication with the stakeholders with a non-technical background [32, 38, 119, 313].
- *Maintenance:* A declarative approach makes possible a *one place representation* of every rule and fact, which is a great advantage when it comes to the maintainability of the specification [241].
- *Knowledge enhancement:* The rules used in an organization, and as such in a supporting CIS, are not always explicitly given. In the words of Stamper [295] "Every organization, in as far as it is organized, acts as though its members were confronting to a set of rules only a *few of which may be explicit* [1]." This has inspired certain researchers to look upon CIS specification as a process of rule reconstruction [108], i.e. the goal is not only to represent and support rules that are already known, but also to uncover de facto and implicit rules which are not yet part of a shared organizational reality, in addition to the construction of new, possibly more appropriate ones.

On the other hand, several problems have been observed when using a simple rule-format. Although addressed in different ways in different areas, many of these also applies to the use of rules for conceptual modeling.

- Every statement must be either true or false, there is nothing in between.
- It is usually not possible to distinguish between rules of necessity and deontic rules [329].
- In many rule modeling languages it is not possible to specify who the rules applies to.

[1] Our italics.

- Formal rule languages have the advantage of eliminating ambiguity. However, this does not mean that rule based models are easy to understand. There are two problems with the comprehension of such models, both the comprehension of single rules, and the comprehension of the whole rule-base. Whereas the traditional operational models have decomposition and modularization facilities which make it possible to view a system at various levels of abstraction and to navigate in a hierarchical structure, rule models are usually *flat*. With many rules such a model soon becomes difficult to grasp, even if each rule should be understandable in itself. According to Li [192] this often makes rule-based systems both unmaintainable and untestable and as such unreliable.
- A general problem is that a set of rules is either consistent or inconsistent. Since human organizations may often have more or less contradictory rules, this is a serious problem.

Some approaches to rule-based modeling that tries to address these problems are presented below. In Chapter 7 we will look upon some examples of how rules can be linked to other modeling languages. Another approach is to relate rules in so-called *goal-oriented approaches*. This also increases the organizational connection by linking rules that are implemented in the CIS to organizational policies and goals, thus addressing the *why*-dimension.

In the ABC method developed by SISU [333] a goal-model is supported, where goals can be said to obstruct, contribute to, or imply other goals. A similar model is part of the F3 modeling languages [40]. Other examples of goal-oriented requirement approaches is reported by Feather [87] where the possible relations between goals and policies are *Supports*, *Impedes*, and *Augments*. Goals can also be *subgoals* i.e decompositions of other goals. Sutcliffe and Maiden [297], and Mylopoulos et al. [232] who use a rule-hierarchy for the representation of non-functional requirements are other examples which we will describe further below.

Sutcliffe: [297] differentiate between six classes of goals:

- 1. Positive state goals: Indicate states which must be achieved.
- 2. Negative state goals: Express a state to be avoided.
- 3. Alternative state goal: The choice of which state applies depends on input during runtime.
- 4. Exception repair goal: In these cases nothing can be done about the state an object achieves, even if it is unsatisfactory and therefore must be corrected in some way.
- 5. Feedback goals: These are associated with a desired state and a range of exceptions that can be tolerated.
- 6. Mixed state goals: A mixture of several of the above.

For each goal-type there is defined heuristics to help refine the different goal-types. Most parent nodes in the hierarchy will have 'and' relations with the child nodes, as two or more sub-goals will support the achievement of an upper level goal, however there may be occasions when 'or' relations are required for alternatives. Goals are divided into policies, functional goals and domain goals. The policy level describes statements of what should be done. The functionally level has linguistic expressions containing some information about how the policy might be achieved. Further relationship types may be added to show goal conflicts, such as 'inhibits', 'promotes', and 'enables' to create an argumentation structure. On the domain level templates are used to encourage addition of facts linking the functional view of aims and purpose to a model in terms of objects, agents, and processes.

Figure 5.11 illustrates a possible goal hierarchy for a library.

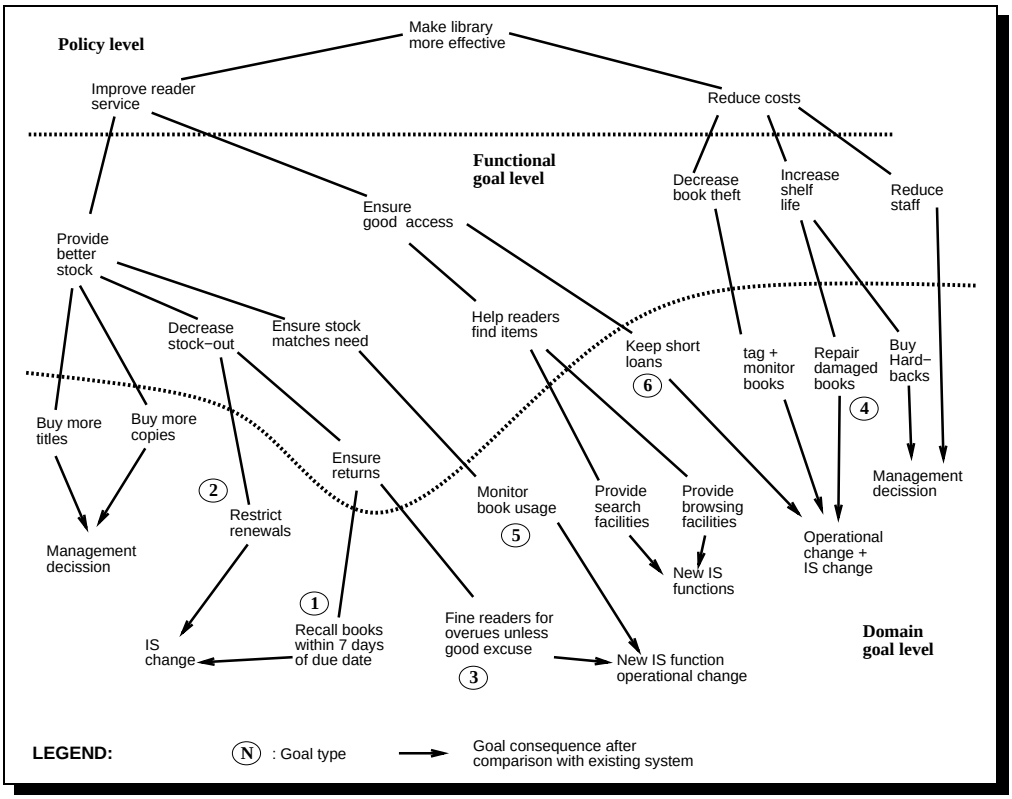

Figure 5.11: Example of a goal hierarchy (From [297])

Mylopoulos et al.: [55, 232] describes a similar structure for representing non-functional requirements. The framework consists of five major components: A set of goals for representing non-functional requirements, design decisions and arguments in support of or against other goals; a set of link types for relating goals and goal relationships; a set of generic methods for refining goals into other goals; a collection of correlation rules for inferring potential interaction among goals; and finally, a labeling procedure which determines the degree to which any given non-functional requirement is being addressed by a set of design decisions. Goals are organized into a graph-structure in the spirit of and/or-trees, where goals are stated in the nodes. The goal structure represents design steps, alternatives, and decisions with respect to non-functional requirements. Goals are of three classes:

- Nonfunctional requirements goals: This includes requirements for accuracy, security, development, operating and hardware costs, and performance.
- Satisficing goals: Design decisions that might be adopted in order to satisfice one or more nonfunctional requirement goal.
- Arguments: Represent formally or informally stated evidence or counter-evidence for other goals or goal-refinements.

Nodes are labeled as undetermined, satisficed and denied.

The following link types are supported describing how the satisficing of the offspring or failure thereof relates to the satisficing of the parent goal:

- *sub*: The satisficing of the offspring contributes to the satisficing of the parent.
- *sup*: The satisficing of the offspring is a sufficient evidence for the satisficing of the parent.
- *-sub*: The satisficing of the offspring contributes to the denial of the parent

- *-sup*: The satisficing of the offspring is a sufficient evidence for the denial of the parent.
- *und*: There is a link between the goal and the offspring, but the effect is as yet undetermined.

Links can relate goals, but also links between links and arguments are possible. Links can be induced by a method or by a correlation rule (see below).

Goals may be refined by the designer, who is then responsible for satisficing not only the goal's offspring, but also the refinement itself represented as a link. Alternatively, the framework provides goal refinement methods which represent generic procedures for refining a goal into one or more offsprings. These are of different kinds: Goal decomposition methods, goal satisficing methods, and argumentation method.

As indicated above, the non-functional requirements set down for a particular system may be contradictory. Guidance is needed in discovering such implicit relationship and in selecting the satisficing goals that best meet the need of the non-functional goals. This is achieved either through external input by the designer or through generic correlation rules.

An example showing how to fulfill the security requirements of a bank's credit card system is given in Figure 5.12.

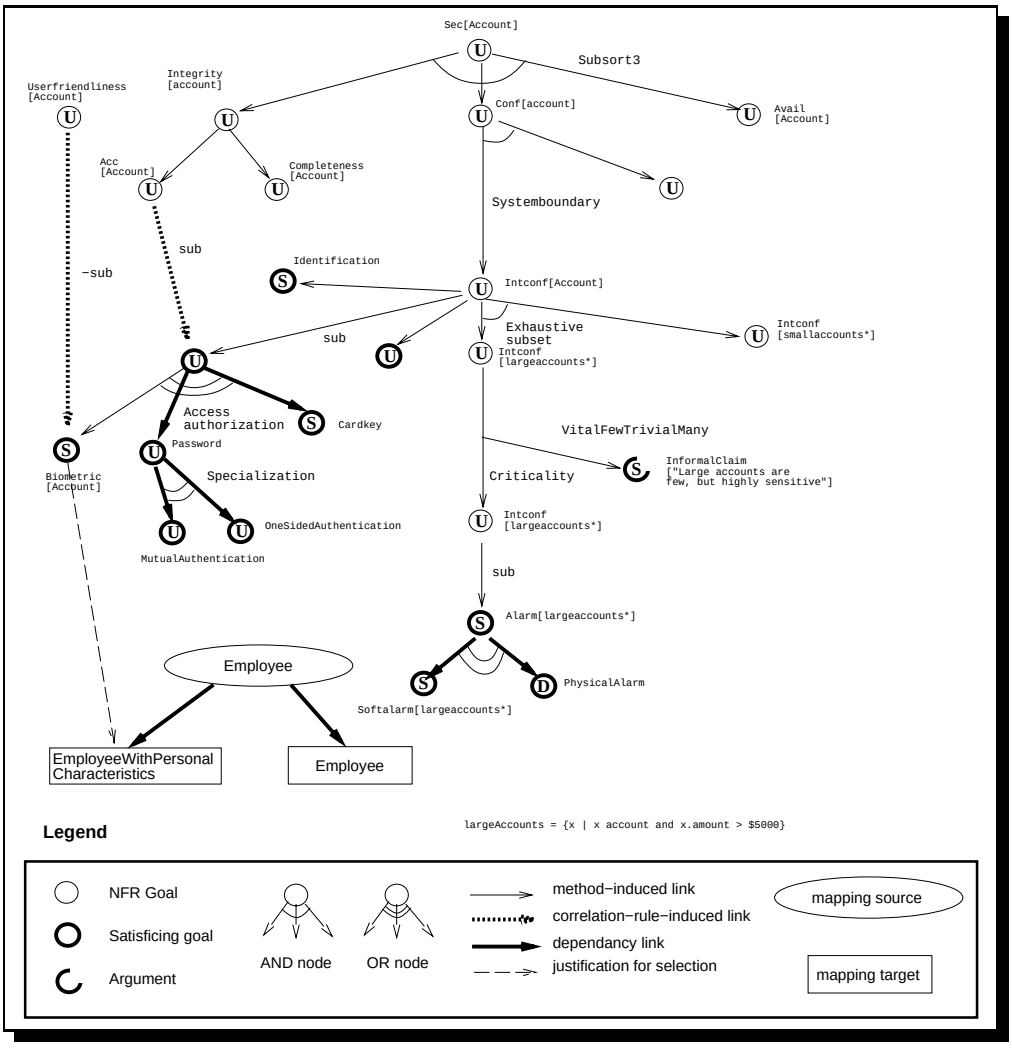

Figure 5.12: Example of a goal-graph (From [55])

Easterbrook [79] illustrates how to model inconsistencies in the same model, thus in a sense implementing goal-hierarchies with or-graphs, and linking the differing views into viewpoints. In Figure 5.13, we see examples of such hierarchies, and also examples of viewpoints, i.e. which views that are currently being compared with each other. The general goal of the librarian is a wish to maximize (improve) circulations. In the first figure, the alternatives being considered are having fixed fines, incremental fines or no fines. In the second 'no fines' is split up into having no sanction, or having sanctions that curtail or restrict borrowing for those not delivering their books in time.

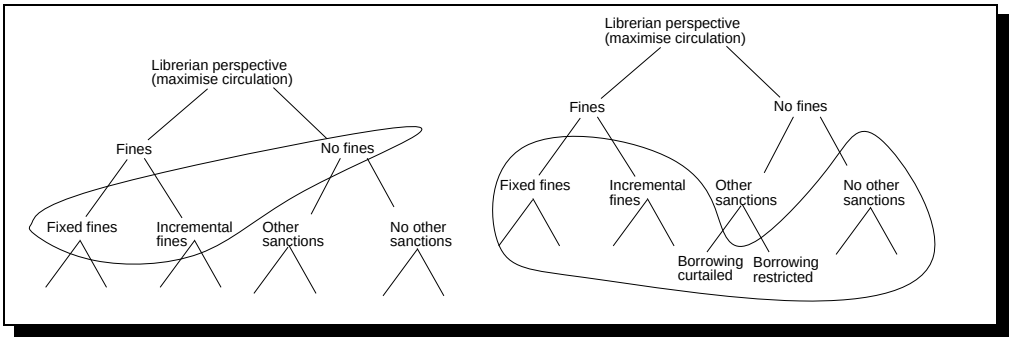

Figure 5.13: Example of viewpoint hierarchies

5.2.5 The Object Perspective

The basic phenomena of object oriented modeling languages are similar to those found in most object oriented programming languages:

- **Object**: An *object* is an "entity" which has a unique and unchangeable identifier and a local state consisting of a collection of attributes with assignable values. The state can only be manipulated with a set of *methods* defined on the object. The value of the state can only be accessed by sending a *message* to the object to call on one of its methods. The details of the methods may not be known, except through their interfaces. The happening of an operation being triggered by receiving a message, is called an *event*.
- **Process**: The *process* of an object, also called the object's *life cycle*, is the trace of the events during the existence time of the object.
- **Class**: A set of objects that share the same definitions of attributes and operations compose an *object class*. A subset of a class, called *subclass*, may have its special attribute and operation definitions, but still share all definitions of its superclass through *inheritance*.

A survey of current object-oriented modeling approaches is given in [332]. According to this, object-oriented analysis should provide several representation of a system to fully specify it:

- Class relationship models: These are similar to ER models.
- Class inheritance models: Similar to generalization hierarchies in semantic data-models.
- Object interaction models: Show message passing between objects
- Object state tables (or models): Follow a state-transition idea.
- User access diagrams: User interface specification.

A general overview of phenomena represented in object-modeling languages is given in Figure 5.14.

We will return to specific aspects of object-oriented modeling in the subsection on the actor and role perspective presented below, in addition to in the overview of combined approaches in

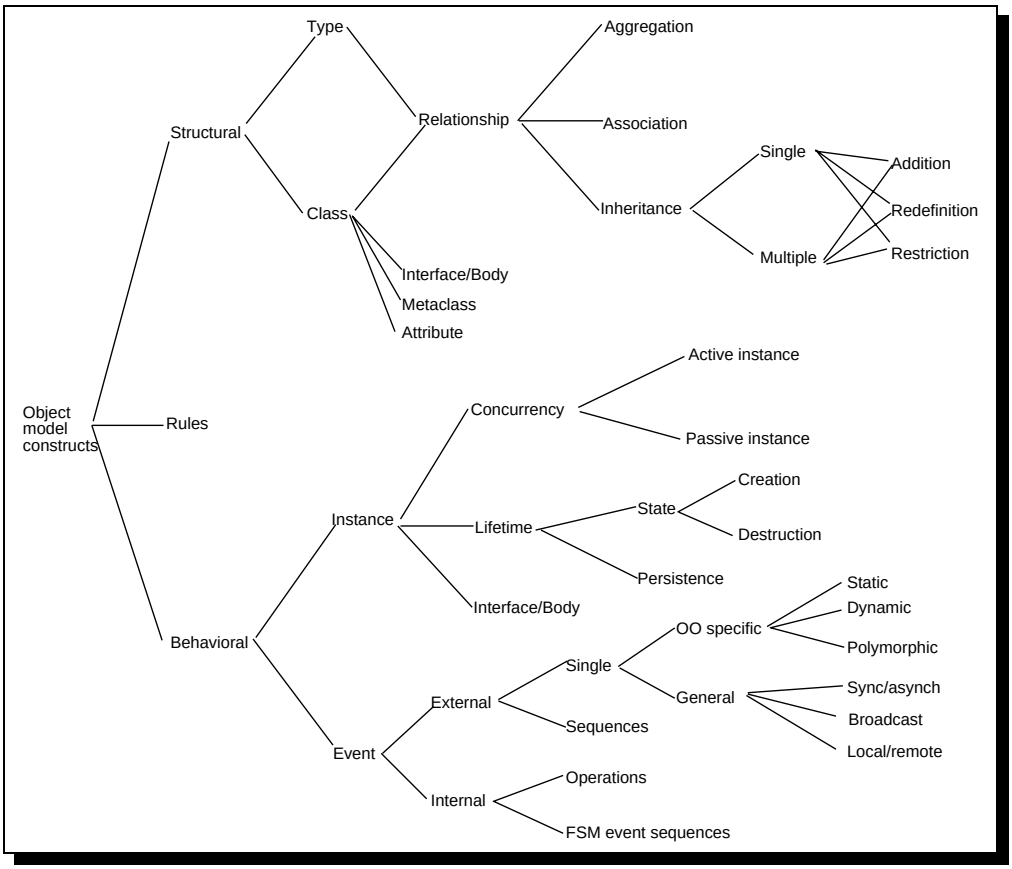

Figure 5.14: General object model(From [332])

Chapter 7.

5.2.6 The Communication Perspective

Much of the work within this perspective is based on language/action theory from philosophical linguistics. The basic assumption of language/action theory is that persons cooperate within work processes through their conversations and through mutual commitments taken within them. *Speech act theory*, which has mainly been developed by Austin and Searle [16, 278, 279] starts from the assumption that the minimal unit of human communication is not a sentence or other expression, but rather the performance of certain kinds of language acts. Illocutionary logic [74, 280] is a logical formalization of the theory and can be used to formally describe the communication structure. The main parts of illocutionary logic is the illocutionary act consisting of three parts, illocutionary context, illocutionary force, and propositional context.

The context of an illocutionary act consist of five elements: Speaker (S), hearer (H), time, location, and circumstances.

The illocutionary force determines the reasons and the goal of the communication. The central element of the illocutionary force is the illocutionary point, and the other elements depend on this. Five illocutionary points are distinguished [279].

- *Assertives*: Commit S to the truth of the expressed proposition (e.g. It is raining).
- *Directives*: Attempts by S to get H to do something (e.g. Close the window).

- *Commissives*: Commit S to some future course of action (e.g. I will be there).
- *Declarations*: The successful performance guarantees the correspondence between the proposition p and the world (e.g. The ball is out).
- *Expressives*: Express the psychological state about a state of affairs specified in the proposition. (e.g. Congratulations!).

Besides the illocutionary point, the illocutionary force contains six more elements:

- Degree of strength of the illocutionary point: Indicates the strength of the direction of fit.
- Mode of achievement: Indicates that some conditions must hold for the illocutionary act to be performed in that way.
- Propositional content conditions: E.g. if a speaker makes a promise, the propositional content must be that the speaker will cause some condition to be true in the future.
- Preparatory condition: There are basically two types of preparatory conditions, those dependant on the illocutionary point and those dependant on the propositional content.
- Sincerity conditions: Every illocutionary act expresses a certain psychological state. If the propositional content of the speech act conforms with the psychological state of the speaker, we say that the illocutionary force is sincerer.
- Degree of strength of sincerity condition: Often related to the degree of strength of the illocutionary point.

Speech acts are elements within larger conversational structures which define the possible courses of action within a conversation between two actors. One class of conversational structures are what Winograd and Flores [337] calls 'conversation for action'. Graphs similar to state transition diagrams have been used to plot the basic course of such a conversation (see Figure 5.15).

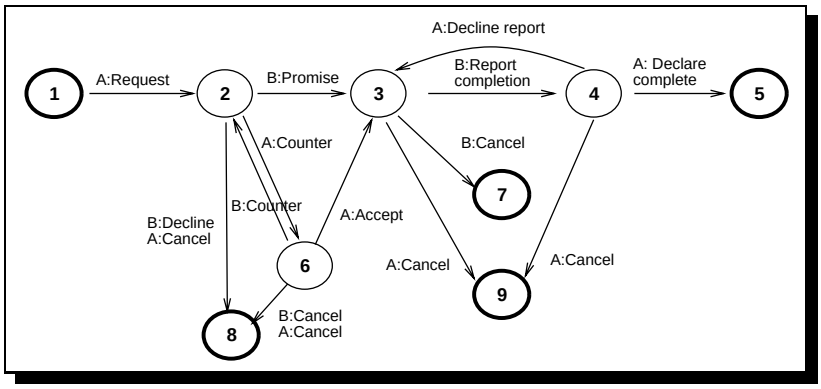

Figure 5.15: Conversation for action (From [337])

This application of speech act-theory forms the basis for several computer systems, the best known being the Coordinator [89].

Speech act theory is also the basis for modeling of work-flow as coordination among people in Action Workflow [221]. The basic structure is shown in Figure 5.16.

Two major roles, customer and supplier, are modeled. Work-flow is defined as coordination between actors having these roles, and is represented by a conversation pattern with four phases. In the first phase the customer makes a request for work, or the supplier makes an offer to the customer. In the second phase, the customer and supplier aims at reaching a mutual agreement about what is to be accomplished. This is reflected in the contract conditions of satisfaction. In the third phase, after the performer has performed what has been agreed upon and completed the work, completion is declared for the customer. In the fourth and final phase the customer assess

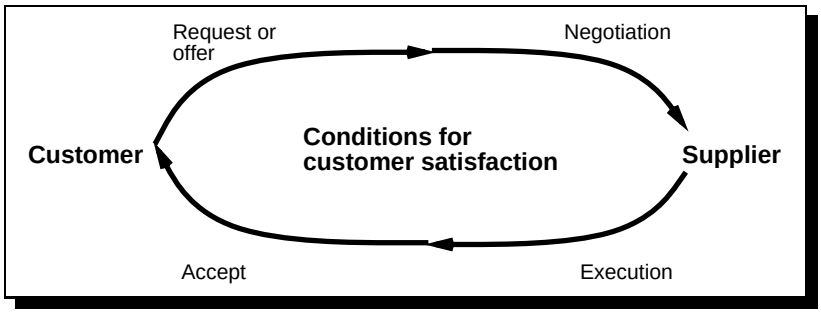

Figure 5.16: Main phases of action workflow

the work according to the conditions of satisfaction and declares satisfaction or dissatisfaction. The ultimate goal of the loop is customer satisfaction. This implies that the work-flow loop have to be closed. It is possible to decompose steps into other loops. The specific activities carried out in order to meet the contract are not modeled.

Habermas took Searle's theory as a starting point for his theory of communicative action[118]. Central to Habermas is the distinction between strategic and communicative action. When involved in strategic action, the participants strive after their own private goals. When they cooperate, they are only motivated empirically do to so: they try to maximize their own profit or minimize their own losses. When involved in communicative action, the participants are oriented towards mutual agreement. The motivation for cooperation is thus rational. In any speech act the speaker S raises three claims: a claim to truth, a claim to justice, and a claim to sincerity. The claim to truth refers to the object world, the claim to justice refers to the social world of the participants, and the claim to sincerity refers to the subjective world of the speaker. This leads to a different classification of speech acts, using the same terms as in [72]:

- Imperative: S aims at a change of the state in the objective world and attempts to let H act in such a way that this change is brought about. The dominant claim is the power claim. Example; " I want you to stop smoking"
- Constativa: S asserts something about the state of affairs in the objective world. The dominate claim is the claim to truth. Example: "It is raining"
- Regulative: S refers to a common social world, in such a way that he tries to establish an interpersonal relation which is considered to be legitimate. The dominant claim is the claim to justice. Example: "Close the window", "I promise to do it tomorrow".
- Expressiva: S refers to his subjective world in such a way that he discloses publicly a lived experience: The dominant claim is the claim to sincerity. Example: "Congratulations" .

A comparisons between Habermas' and Searle's classifications is given in Figure 5.17.

In addition to the approach to workflow-modeling described above, several other approaches to conceptual modeling are inspired by the theories of Habermas and Searle. We will mention one here, ABC. Another, SAMPO, is presented in Chapter 7.3.

ABC-diagrams: Dietz [71] differentiate between two kinds of conversations:

- Actagenic, where the result of the conversation is the creation of something to be done (agendum), consisting of a directive and a commissive speech act.
- Factagenic, which are conversation which are aimed at the creation of facts typically consisting of a assertive and a declarative act.

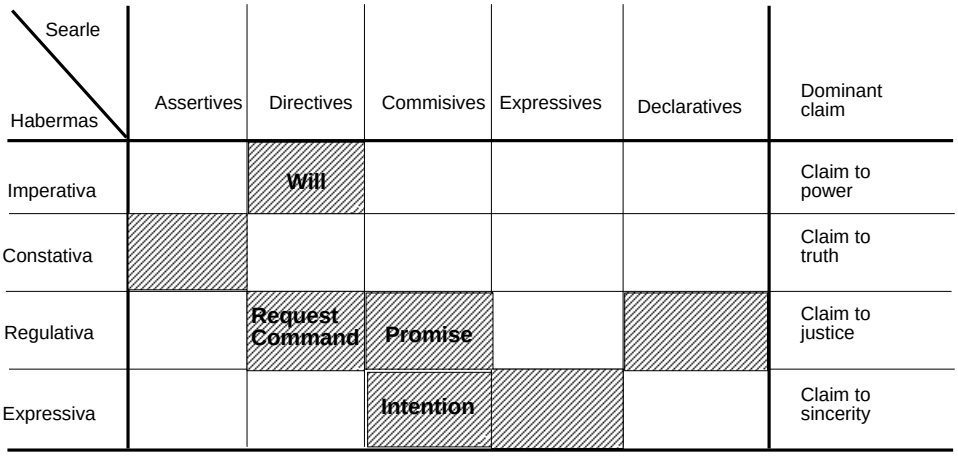

Figure 5.17: Comparing communicative action in Habermas and Searle (From [72])

Actagenic and factagenic conversations are both called performative conversations. Opposed to these are informative conversations where the outcome is a production of already created data. This include the deduction of data using e.g. derivation rules.

An agendum is a pair $< a, p >$ where a is the action to be executed and p the period in which this execution has to take place.

In the factagenic conversation, the result of the execution are stated by the supplier. It is successful if the customer accepts these results. Note the similarities between this and the workflow-loop in action workflow.

In order to concentrate on the functions performed by the subjects while abstracting from the particular subjects that perform a function, the notion of actor is introduced. An actor is defined by the set of actions and communications it is able to perform.

An actor that is element of the composition of the subject system is called an internal actor, whereas an actor that belongs to the environment is called an external actor. Transaction types of which the initiator as well as the executor is an internal actor is called an internal transaction. If both are external, the transaction is called external. If only one of the actors is external it is called an interface transaction type. Interaction between two actors takes place if one of them is the initiator and the other one is the executor of the same transaction type. *Interstriction* takes place when already created data or statuses of current transactions are taken into account in carrying out a transaction.

In order to represent interaction and interstriction between the actors of a system, Dietz introduce ABC-diagrams. The graphical elements in this language are showed in Figure 5.18. An actor is represented by a box, identified by a number. A transaction type is represented by a disk. The operational interpretation of a disk is a store for the statuses through which the transaction of that type pass in the course of time. The disk symbol is called a channel. The diamond symbol is called a bank, and contain the data created through the transaction. The actor who is the initiator of a transaction type is connected to the transaction channel by a generate link (g-link) symbolized by a plain link. The actor who is the executor is connected to the transaction by an execute link (e-link). Informative conversations are represented by inspect links (i-links), symbolized by dashed lines.

In [314] it is in addition illustrated how to show the sequence between transactions in a transaction sequence graph. It is also developed a transaction process model which is an extension

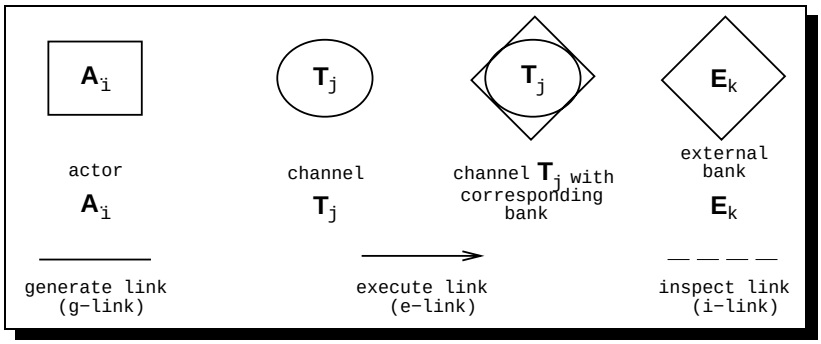

Figure 5.18: The symbols of the ABC-language (From [71])

of the model presented in Figure 5.15 also including an indication of the dominant claim that is potentially countered.

5.2.7 The Actor and Role Perspective

The background for modeling of the kind described here comes both from work on (object-oriented) programming languages (e.g actor-languages [310]), and work on intelligent agents in artificial intelligence (e.g [99, 284]). Note that the terms actor, agent, and role are used differently in the literature. We will use the definitions from Chapter 2.1 if possible.

ALBERT

ALBERT (Agent-oriented Language for Building and Eliciting Real-Time requirements) [78] have a set of specification language for modeling complex real-time cooperative distributed systems which are based on describing a system as a society of agents, each of them with their own responsibilities with respect to the actions happening in the system and its time-varying perception of the behavior of the other agents. A variety of requirements can be described with ALBERT, such as structural, temporal, functional, behavioral, in addition to real-time and cooperative aspects which are covered through the modeling of distributed systems in terms of agents, each of them characterized with time-varying communication possibilities. Communication mechanisms allow to describe how an agent perceive data made available to it by other agents and show parts of its data to other agents. We will here concentrate on the agent modeling.

Agents, as defined in ALBERT, may be seen as a specialization of objects. Models are made at two levels.

- Agent level: A set of possible behaviors are associated with each agent without any regard to the behavior of other agents
- Society level: Interactions between agents are taken into account and lead to additional restrictions on the behavior of each individual agent.

The formal language is based on a variant of temporal logic extended with actions, agents, and typical patterns of constraints. The declaration of agents consist in the description of the state structure and the list of the actions its history can be made of. The state is defined by its components which can be individuals collections of individuals. Components can be time-varying or constant. Agents include a key mechanism that allows the identification of the different instances. A type is automatically associated to each class of agents. Figure 5.19 shows the model associated

with the declaration of the state structure of a cell (a part of a CIM production system).

Sets and instances are depicted as small rectangles with rectangles inside indicating the type. Actions are depicted as small rectangles with ovals inside. Actions might have arguments. A wavy line between components expresses that the value of a component may be derived from others. It is possible to distinguish between internal and external action and to express the visibility relationships linking the agent to the environment. The components within the parallelogram is under the control of the described agent while information outside denotes elements which are imported from other agents of the society the agent belongs to. Boxes within the parallelogram with an arrow going out from them denote that data is exported to the outside.

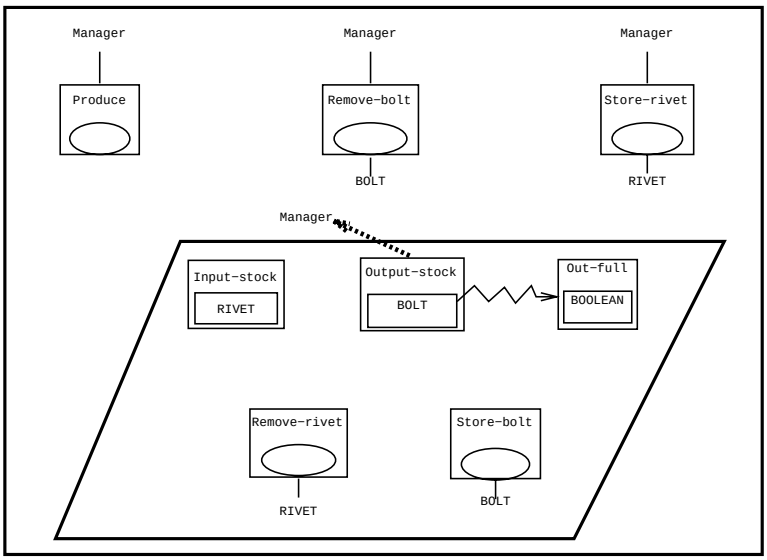

Figure 5.19: Example of an ALBERT model (From [78])

Agents are grouped into societies, which themselves can be grouped into other societies. The existing hierarchy of agents are expressed in term of two combinations: Cartesian product and set. Constraints are used for pruning the infinite set of possible lives of an agent. These are divided into ten headings and three families to provide methodological guidance. The families are:

- Basic constraints: Used to describe the initial state of an agent, and to give the derivation rules for derived components.
- Local constraints: Related to the internal behavior of the agent.
- Cooperative constraints: Specifies how the agent interacts with its environment.

Organizational modeling:

Yu and Mylopoulos [343, 344] have proposed a set of integrated languages to be used for organizational modeling:

- The Actor Dependency modeling language.
- The Agents-Roles-Positions modeling language.
- The Issue-Argumentation modeling language.

The Issue-Argumentation modeling language is an application of a subset of the non-functional framework presented in Section 5.2.4 for organization modeling. The two other modeling languages are presented below.

In actor dependency models each node represent a social actor/role. Each link between the nodes indicates that a social actor depends on the other to achieve a goal. The depending actor is called the *depender*, and the actor that is depended upon is called the *dependee*. The object assigned to each link is called a *dependum*. It is distinguished between four types of dependencies:

- Goal dependency: The depender depends on the dependee to bring about a certain situation. The dependee is expected to make whatever decisions are necessary to achieve the goal.
- Task dependency: The depender depends on the dependee to carry out an activity. A task dependency specifies how, and not why the task is performed.
- Resource dependency: The depender depends on the dependee for the availability of some resources (material or data).
- Soft-goal dependencies: Similar to a goal dependency, except that the condition to be attained is not accurately defined.

The language allows dependencies of different strength: Open, Committed, and Critical. An activity description, with attributes as input and output, sub-activities and pre and post-conditions expresses the rules of the situation. In addition to this is a goal attribute is added to activities. Several activities might match a goal, thus subgoals are allowed. Figure 5.20 gives an example of an actor dependency model.

The Agents-Roles-Positions modeling language similarly consists of a set nodes and links as illustrated in Figure 5.21. An *actor* is here as above used to refer to any unit to which intentional dependencies can be ascribed. The term *social actor* is used to emphasize that the actor is made up of a complex network of associated agents, roles, and positions. A *role* is an abstract characterization of the behavior of a social actor within some specialized context or domain. A *position* is an abstract place-holder that mediates between agents and roles. It is a collection of roles that are to be played by the same agent. An *agent* refers to those aspects of a social actor that are closely tied to its being a concrete, physically embodied individual. We have used the original terms here, although they differ from the terms in our terminology.

Agents, roles, and positions are associated to each other via links: An agent can *occupy* a position, a position is said to *cover* a role, and an agent is said to *play* a role. In general these associations may be many-to-many. An *interdependency* is a less detailed way of indicating the dependency between two actors. Each of the three kinds of actors- agents, roles, and positions, can have sub-parts.

OORASS - Object oriented role analysis, synthesis and structuring

OORASS [267] is really a pure object-oriented method, but we have chosen to present it here since what is special to OORASS is the modeling of roles. A role model can roughly be described as a purification the collaboration graphs introduced in RDD [338]. A role model is a model of object interaction described by means of message passing between roles. It focus on describing patterns of interaction without connecting the interaction to particular objects.

The main parts of a role model is described in Figure 5.22. A *role* is defined as the why-abstraction. Why is an object included in the structure of collaborating objects? What is its position in the organization, what are the responsibilities and duties? All objects having the same position in the structure of objects play the same role. A role only has meaning as a part of some structure. This makes the role different from objects which are entities existing “in their own right”. An object has identity and is thus unique, a role may be played by any number of objects (of any type). An object is also able to play many different roles. In the figure there are two roles A and B. A path between two roles means that a role may ’know about’ the other role so that it

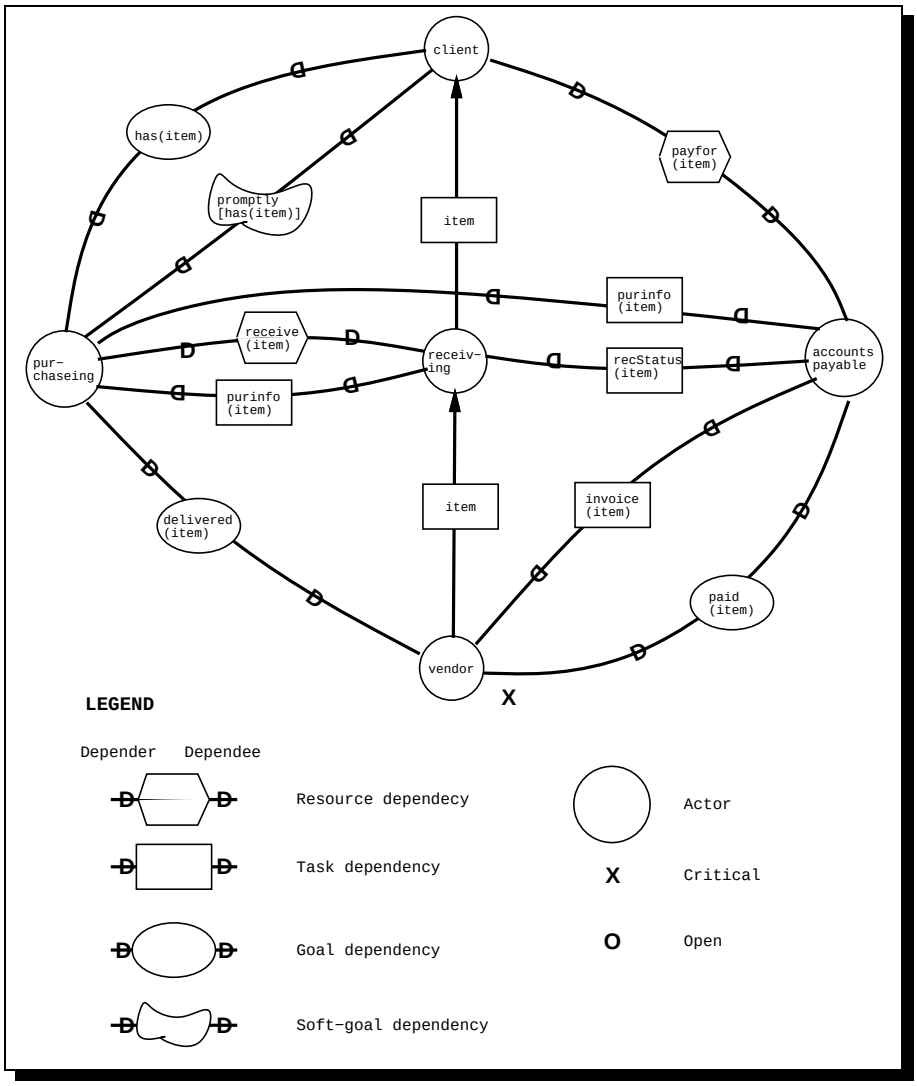

Figure 5.20: Example of an actor dependency model (From [344])

can send messages to it. A path is terminated by a port symbol at both ends. A port symbol may be a single small circle, a double circle, or nothing. Nothing means that the near role do not know about the far role. A single circle (p) indicates that an instance of the near role (A) knows about none or one instance of the far role (B). A double circle (q) indicates that an instance of the near role knows about none, one or more instances of the far role. In the figure 'p' is a reference to some object playing the role B. Which object this is may change during the lifetime of A. If some object is present, we are always assured that it is capable of playing the role B. For a port, one can define an associated set of operations called a contract. These operations is the ones that the near role requires from the far role, not what the near role implements. The signatures offered must be deduced from what is required in the other end.

Role models may be viewed through different *views.*

- Environment view: The observer can observe the system interact with its environment.
- External view: The observer can observe the messages flowing between the roles.
- Internal view: The observer can observe the implementation

Other views are given using additional languages with structural, functional, and behavioral per-

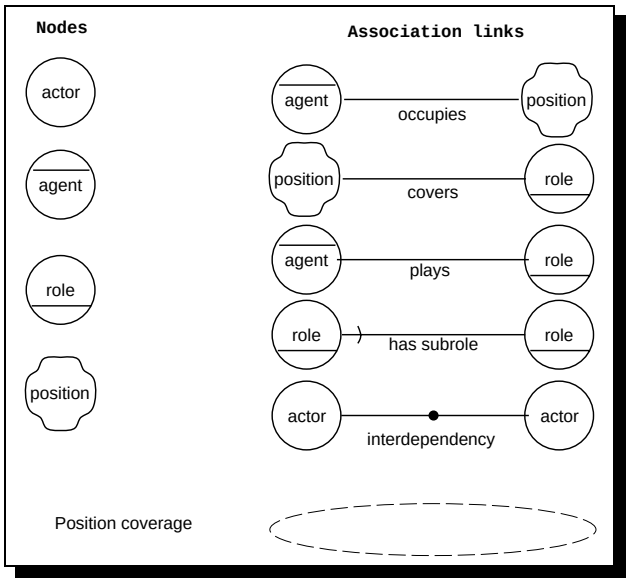

Figure 5.21: Symbols in agents-role-position modeling language (From [343])

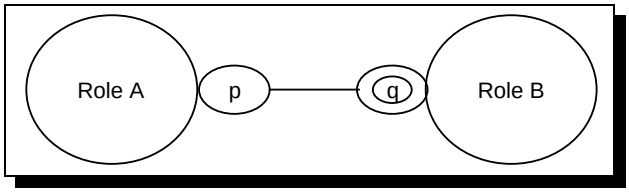

Figure 5.22: Symbols in the OORASS role interaction language

spectives.

5.3 On Conceptual Relativism

We have above presented different perspectives towards conceptual modeling. Based on social construction theory, the general features of the world can not be said to exist a priori. According to this belief one might wish go to the other extreme — an approach without any presumptions at all. However, this is impossible. Any methodology and any language implies some presumptions. Thus, having an approach totally free of presumptions would mean to have no approach at all, inventing a new one fit for the specific problem for every new development and maintenance task. For philosophers this might be acceptable, but engineers are expected to adapt to certain demands for efficiency. Inventing a new approach for every development and maintenance effort would not give us that efficiency, neither is it likely that it will give better CIS-support for the organization. Developing and maintaining a CIS without any fixed ideas about how it should be done would be tedious and unsystematic — as stated in [29] the ad hoc methods used in the earliest days of software development were much worse than the those used today. So clearly one need to make some presumptions, one need to have some fixed ideas. What is necessary is to find a *point of balance* — making enough presumptions for the approach to be systematic and efficient, but not so many that its flexibility and applicability is severely reduced. We can become aware of some

of our presumptions, and in that way emancipate ourselves from some of the limits they place on our thinking, but we can never be free of all presumptions.

As we have illustrated in this chapter, there are a number of different approaches to conceptual modeling each emphasizing different aspects of the perceived reality. Several researchers have claimed that one perspective is better, or more natural, than others:

- Sowa [294] bases his language for conceptual graphs on work on human perception and thinking done in cognitive psychology, and uses this to motivate for the use of the language. We think it is safe to say that even with his convincing discussion, conceptual graphs have had a very limited influence on conceptual modeling practices and the development and maintenance of CISs in most organizations, even if its has received much attention within computer science research [2].
- In the last years, many authors have advocated object-oriented modeling partly based on the claim that it is a more natural way to perceive the world [332]. The view that object-orientation is a suitable abstraction for all situations have been criticized by many lately see e.g. [36, 137, 146]. The report on the First International Symposium on Requirements Engineering [146] said it so strongly that "requirements are not object-oriented. Panelist reported that users do not find it natural to express their requirements in object-oriented fashion". Even if there are cases where an object-oriented perspective is beneficial, it seems not to be an appropriate way of describing all sorts of problems, as discussed in [137]. Newer approaches to OOA claim to attack some of these problem see e.g. [84].
- In Tempora [304], rules were originally given a similar role in that it was claimed that "end users perceive large parts of a business in terms of policies or rules". This is a truth with modification. Even if people may act according to rules, they are not necessarily looking upon it as they are as discussed by Stamper [295]. Rule-based approaches also have to deal with several deficiencies, as discussed earlier in the chapter.
- Much of the existing work on conceptual modeling that has been based on an constructivistic world-view has suggested language/action modeling as a possible cornerstone of conceptual modeling [107, 164, 337], claiming that it is more suitable than traditional "objectivistic" conceptual modeling. On the other hand the use of this perspective has also been severely criticized, also from people sharing a basic constructivistic outlook. An overview of the critique is given in [65]:
  - Speech act theory is wrong in that it assumes a one-to-one mapping between utterances and illocutionary acts, which is not recognizable in real life conversations.
  - The normative use of the illocutionary force of utterances is the basis for developing tools for the discipline and control over organizations member's actions and not supporting cooperative work among equals.
  - The language/action perspective does not recognize that embedded in any conversation is a process for negotiating the agreement of meaning.
  - The language/action perspective misses the locality and situatedness of conversations, because it proposes a set of fixed models of conversations for any group without supporting its ability to design its own conversation models.
  - The language/action perspective offers only a partial insight; it has to be integrated with other theories.
- As discussed above also functionally and structurally oriented approaches have been criticized in the literature [39, 245].

[2] The third international conference on the topic was held in August 1995.

Although the use of a perspective has been criticized, this does not mean that modeling according to a perspective should be abandoned, as long as we do not limit ourselves to one single perspective. A model expressed in a given language emphasize a specific way of ordering and abstracting ones internal reality. One model in a given language will thus seldom be sufficient. With this in mind more and more approaches are based on the combination of several modeling languages. There are at least four general ways of attacking this:

- 1. Use existing single-perspective languages as they are defined, without trying to integrate them further. This is the approach followed in many existing CASE-tools.
- 2. Refine common approaches to make a set of formally integrated, but still partly independent set of languages.
- 3. Develop a set of entirely new integrated conceptual modeling languages.
- 4. Create frameworks that can be used for creating the modeling languages that are deemed necessary in any given situation.

Due to the increased possibilities of consistency checking and traceability across specifications, in addition to better possibilities for the conceptual models to serve as input for code-generation, and to support validation techniques such as execution, explanation generation, and animation the second of these approaches has been receiving increased interest, especially in the academic world. Basing the modeling languages on well-known modeling languages also have other advantages on behalf of perceptibility, and because of the existing practical experience with these languages. Also many examples of the third solution exist, e.g. ARIES [152] and DAIDA [148], and of the fourth e.g. [240, 246] and work on so-called meta-CASE systems e.g. [209, 316]. Work based on language-modeling might also be used to improve the applicability of approaches of all the other types.

5.4 Chapter Summary

We have in this chapter presented an introduction to the field of conceptual modeling.

A discussion of the applicability of conceptual modeling in the light of social construction theory is given, and based on this it is stated how conceptual modeling can be looked upon as a process of social construction.

Many perspectives can be used for conceptual modeling, and we have in this chapter presented examples of modeling languages which emphasize different perspectives that can be identified in conceptual modeling language literature. The presented perspectives were the structural, functional, behavioral, rule, object, communication, and actor/role perspectives. All of these have received critique, and it is our belief that to make conceptual modeling useful one should apply a set of integrated modeling languages supporting the modeling along different perspectives, being supported by a language modeling approach for extensibility.

We will in the next chapter investigate into the *quality* of conceptual models and modeling languages, freeing ourselves from concerning specific perspectives for a moment. In Chapter 7 we give an overview over some approaches to conceptual modeling which combine several perspectives.

Chapter 6

Quality of Conceptual Models

We will in this chapter describe a framework for understanding quality in conceptual modeling. The proposed framework will be applied when comparing different approaches to conceptual modeling on their potential to produce high-quality models.

Previous proposals for quality goals for conceptual models as summarized in [63] have included many useful aspects, but unfortunately in the form of unsystematic lists as discussed in [196, 199]. They are also often restricted in the kind of models they regard (e.g. requirements specifications [58]) or the modeling language [225]. Some recent frameworks [195, 199, 261] have attempted to take a more structured approach to understand quality. We will briefly present the main parts of these frameworks, before evaluating them and proposing an extended framework.

6.1 Overview and Evaluation of Existing Frameworks

We will here give a short overview of three existing frameworks:

- Lindland et al. [199].
- NATURE [261].
- The semiotic ladder in FRISCO [195].

The two latter is compared against the framework of Lindland et al. [199]. Based on this comparison, we will present our extended framework which is based on this.

6.1.1 Lindland/Sindre/Sølvberg's Framework

The main structure of this framework is illustrated in Figure 6.1. The basic idea is to evaluate the quality of models along three dimensions — syntax, semantics, and pragmatics — by comparing sets of statements. These sets are:

- $\mathcal{M}$, the model, i.e. the set of all the statements explicitly or implicitly made in the model.
- $\mathcal{L}$, the language extension, i.e. the set of all statements which are possible to make according to the vocabulary and grammar of the modeling languages used.
- $\mathcal{D}$, the modeling domain, i.e. the set of all statements which would be correct and relevant about the problem at hand.
- $\mathcal{I}$, the audience interpretation, i.e. the set of all statements which the audience (i.e. various actors in the modeling process) think that the model consists of.

Model quality is defined using the relationships between the model and the three other sets:

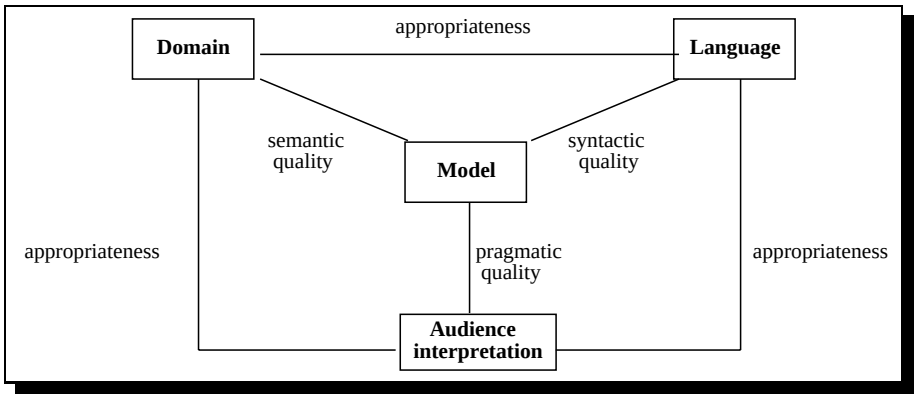

Figure 6.1: The framework for model quality by Lindland et al. (From [199])

- *syntactic quality* is the degree of correspondence between model and language extension, i.e. the set of syntactic errors is $\mathcal{M} \setminus \mathcal{L}$.
- *semantic quality* is the degree of correspondence between model and domain. If $\mathcal{M} \setminus \mathcal{D} \neq \emptyset$ the model contains invalid statements; if $\mathcal{D} \setminus \mathcal{M} \neq \emptyset$ the model is incomplete. Since total validity and completeness are generally impossible, the notions of *feasible validity* and *feasible completeness* were introduced. Feasible validity is reached when the benefits of removing invalid statement from $\mathcal{M}$ are less than the drawbacks, whereas feasible completeness is reached when the benefits of adding new statements to $\mathcal{M}$ is less than the drawbacks. The term drawback is used instead of the more familiar term cost in an effort to cover both economic issues and factors like user preferences and ethics.
- *pragmatic quality* is the degree of correspondence between model and audience interpretation (i.e., the degree to which the model has been understood). If $\mathcal{I} \neq \mathcal{M}$, the comprehension of the model is not completely correct. Usually, it is neither necessary nor possible that the whole audience understand the entire conceptual model — instead each group in the audience should understand the part of the model which is relevant to them. *Feasible comprehension* was defined along the same lines as feasibility for validity and completeness.

In addition to these primary quality concerns, it is pointed out that correspondence between domain and language, between domain and audience interpretation, and between language and audience interpretation may affect the model quality indirectly. These relationships are all denoted *appropriateness* as shown in Figure 6.1. For more details on this framework, the reader should consult [199]. The parts of the framework dealing with fault detection has been applied in connection with integrating the development and testing of object-oriented systems in [219].

6.1.2 Pohl's Framework

Pohl's framework [261] which is part of the NATURE-project [147] is summarized in Figure 6.2. In this framework the requirements specification process, which often includes conceptual modeling, is stretched out along three dimensions:

- *The specification dimension* deals with the degree of requirements understanding. At the beginning of the process, this understanding is opaque. The desired output of the requirement specification process is a complete CIS-specification, where completeness is measured against some standard, guideline, or another model.

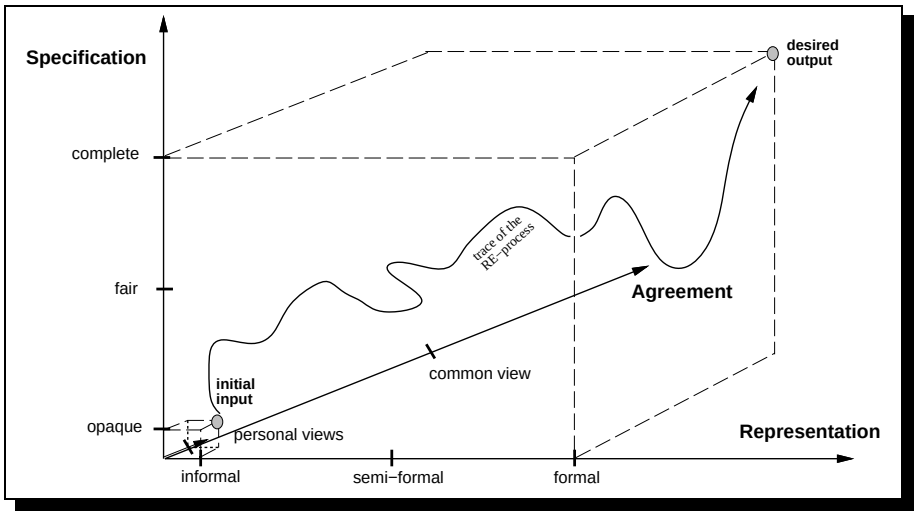

Figure 6.2: Pohl's framework for the requirement specification process (From [261])

- *The representation dimension* deals with the degree of formality. Various languages can be used in the process; informal ones such as natural language, semi-formal ones such as many diagrammatical modeling languages, and formal ones (e.g., logic). At the beginning of the process, statements will usually be informal. Since formal representations allow reasoning and partial code-generation, these are more product-oriented. Hence, a transformation of informal requirements to a formal representation is regarded to be desirable.
- *The agreement dimension* deals with the degree of agreement. The requirement specification process has many stakeholders, and in the beginning each of these will have their personal views concerning the requirements to be made. The goal of the process is to reach agreement on the requirements. Detected conflicts must be solved through discussions among those affected.

6.1.3 The Use of the Semiotic Ladder in FRISCO

The FRISCO report [195] identifies that the means of communication and related areas can be examined in a semiotic framework. The below semiotic layers for communication are distinguished, forming a semiotic ladder. Together with the description is listed a number of illustrative words, which are examples of aspects often treated at the level in question.

- *Physical:* This layer concentrate on the physical appearance, the media and amount of contact available. Examples are signals, traces, hardware, component density, and speed.
- *Empirical:* This layer concentrate on aspects such as the entropy, variety and equivocation encountered. Examples are pattern, variety, noise, entropy, channel capacity, codes, efficiency, and redundancy.
- *Syntactic:* This layer looks on the language, the structure and logic used. Examples are formal structure, language, logic, data, records, files, and software.
- *Semantic:* The meanings and validity of what is expressed is covered on this layer. Example are meaning, propositions, validity, truth, and signification.
- *Pragmatic:* The pragmatic layer concentrate on the intentions and signification behind the expressed statements. Examples are intentions, communication, conversation, and negotiation.

- *Social:* Finally this layer discuss the interests, beliefs, and commitments shared as a result of the communicative process. Examples are beliefs, expectations, commitments, contracts, laws, and culture.

These layers can be divided into two groups in order to reveal the technical vs. the social aspect. Physics plus empirics plus syntactics comprise an area where technical and formal methods are adequate. However, semantics plus pragmatics plus the social sphere cannot be explored using those methods unmodified. This indicates than one has to include human judgement when discussing aspects on the higher semiotic levels.

6.1.4 Overall Comparison

Although the frameworks of Lindland et al. and Pohl have a quite different appearance, they are rather similar in their deeper structure. The following observations can be made:

- The *representation* dimension corresponds to the *syntactic* dimension, since both these deal with the relationship between the specification and the language(s) used. The main differences in this respect is that Pohl's framework discusses several languages, whereas Lindland et al.'s framework sees the language as one and just considers whether the specification is correct according to the rules of that language (which may be a union of several languages, formal, semi-formal, and informal). It should also be noted that Pohl's framework regards a formal specification as a *goal*. Lindland's framework states that formality is a *mean* to reach a syntactically correct specification, as well as higher semantic and pragmatic quality through consistency checking and model executions of different kinds.
- The *specification* dimension corresponds to the *semantic* dimension, since both these deal with the goal of completeness. A notable difference here is that Pohl sees completeness as the sole goal (possibly including validity?), whereas Lindland's framework also identifies the notions of validity and feasibility. The reason for this discrepancy seems to be a somewhat different use of the term completeness, where Pohl uses the term relative to a standard, whereas Lindland et al. uses it relative to the set of all statements which would be correct and relevant about the problem at hand.
- The *agreement* dimension is related to the *pragmatic* dimension, since both these deal with the specification's relationship to the involved audience. The difference is that Pohl states the goal that the specification should be agreed upon, whereas Lindland et al. aim at letting the model be understood. These goals are clearly related. Agreement without understanding is not very useful in a democratic process. On the other hand, using the semiotic layers of FRISCO, it is more appropriate to put agreement into the social realm going beyond the framework of Lindland et al.

Comparing the Lindland framework with FRISCO, we see that the framework suggested by Lindland et al. to some extend take the insight of semiotic layers into account by differentiating between syntactic, semantic, and pragmatic quality. Even if the terms are used somewhat differently, the overall layers coincide. On the other hand, neither the lower physical and empirical layer or the social layer can be said to be covered in the existing framework. As also indicated above, the social aspects of agreement is currently not handled in a satisfactory way. When discussing agreement, the term 'domain' as currently used is also problematic, since it represents an ideal knowledge about a particular situation, a knowledge not obtainable for the actors of the audience that are to agree.

6.1.5 Critique of the Framework of Lindland et al.

Based on the above discussion, the role of conceptual models as described in Chapter 5, and our philosophical outlook, we can identify the following weaknesses with the framework described by Lindland et al:

- The notion of 'domain' is problematic seen from a constructivistic viewpoint, since it seems to imply the existence of an objectively, true solution. Even when only used as a conceptual fixpoint to be able to define quality terminology, it misses the important factor of the differing explicit knowledge of the participants in the modeling process. This also bring up the point of knowledge quality, i.e. how to select participants to the modeling effort among a potentially large set of stakeholders.
- Physical and empirical aspects of conceptual models are not discussed.
- Social aspects such as agreement are not discussed.

Based on this we will present an extension of the framework of Lindland et al. performed by us in cooperation with the original authors taking the above critique into account. All aspects of the extension which are based on a constructivistic view are original contributions by the author. The new framework has earlier been presented in [177, 178]. In addition will we include a discussion of language quality based on [282, 286], adapted to the overall framework which together with knowledge quality look in more detail on the aspects which in the original framework were termed 'appropriateness'.

6.2 A Framework for Quality of Conceptual Models

We will first outline the overall framework. We will then look upon quality on different semiotic levels and includes physical, syntactical, semantical, pragmatic, and social quality in addition to knowledge and language quality. Empirical aspects are treated under the discussion on pragmatic quality. The main concepts and their relationships are shown in Figure 6.3. Primary quality goals for a conceptual model are indicated in the figure with solid lines, whereas the relationships covering the appropriateness-relationships of the original framework are showed using dotted lines. As in Lindland et al, we take a set-theoretic approach to the discussion of the different quality aspects. Sets are written using $\mathcal{CALIGRAPHIC}$ letters, whereas elements of sets are written in ordinary uppercase letters. A list of the symbols used can be found in Appendix G. People familiar with the field of logic programming should be aware of that the terminology we apply use many terms differently from how they are used in that field.

- $\mathcal{A}$, the audience, i.e. the union of the set of individual actors A_1,...,A_k the set of organizational actors A_{k+1},...,A_n and the set of technical actors A_{n+1},...,A_m who needs to relate to the model. The individuals being members of the audience are called the *participants* of the modeling process. The participants $\mathcal{P}$ is a subset of the set of stakeholders $\mathcal{S}$ of the process of creating the model. A technical actor is typically a computer program e.g. a CASE-tool, which must "understand" parts of the model to automatically manipulate it to for instance perform code-generation or execution based on the conceptual model.
  The audience often change during modeling, some persons leaving the project as it takes place, whereas others join the project after it has started.
- $\mathcal{L}$, the language extension, i.e. the set of all statements that are possible to make according to the vocabulary and syntax of the modeling languages used. Several languages can be used in the same modeling effort, corresponding to the sets $\mathcal{L}_1$,...,$\mathcal{L}_j$. These languages can

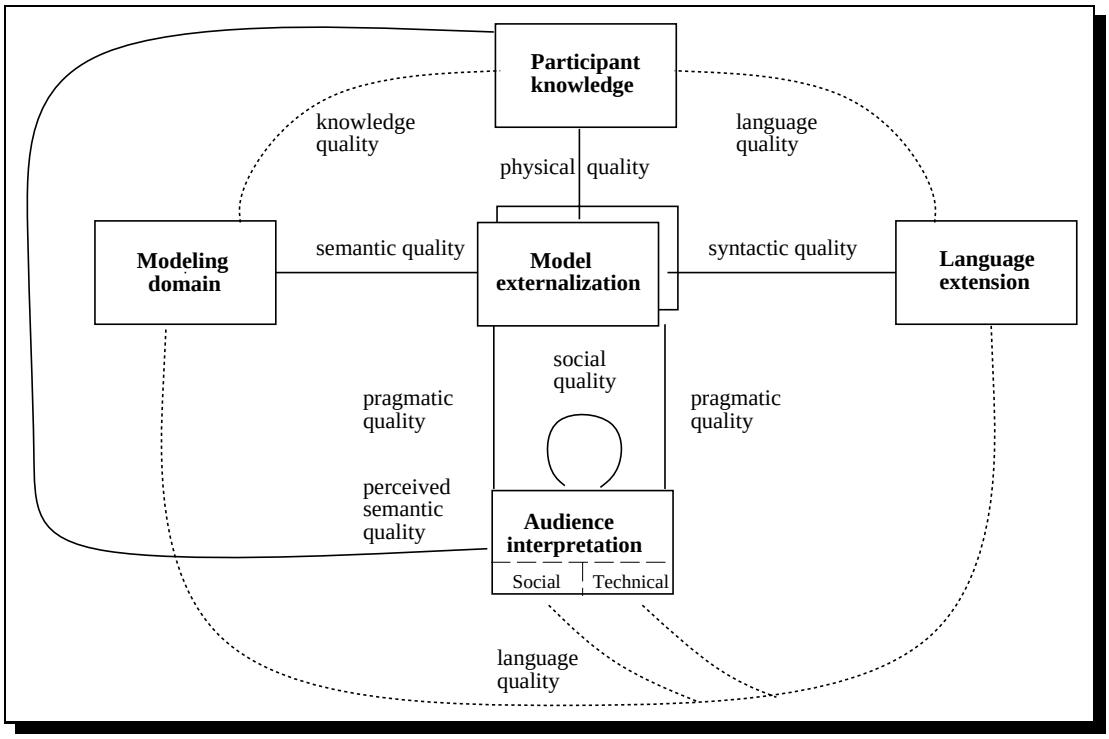

Figure 6.3: Extended framework for discussing quality of conceptual models

be inter-related. Sub-languages are related to the complete language by limitations on the vocabulary or on the set of allowed grammar rules in the syntax or both. The statements in the language model of a formal or semi-formal language L_i are denoted with $\mathcal{M}(L_i)$.
$\mathcal{L}$ can be divided into $\mathcal{L}_I$, $\mathcal{L}_S$, and $\mathcal{L}_F$ for statements made in informal, semi-formal and formal (operational) parts of the language, respectively. $\mathcal{L} = \mathcal{L}_I \cup \mathcal{L}_S \cup \mathcal{L}_F$. $\mathcal{L}_L$ denotes the statements with logical semantics.
The languages used in a modeling effort are usually predefined, but one can also create specific modeling languages using e.g. a meta-CASE tool for the modeling effort, in which case the syntax and semantics of the languages have to be inter-subjectivly agreed among the audience as part of the modeling. If one are using an existing language, the "correct" syntax and semantics of the language will be regarded as predefined. One can also choose to apply only parts of the predefined modeling languages, and change this as one go along.

- $\mathcal{M}$, the externalized model, i.e. the set of all statements in someones model of part of the perceived reality written in a language. $\mathcal{M}_E$ is the set of explicit statements in the models, whereas $\mathcal{M}_I$ is the set of implicit statements, being the statements not made, but implied through the deduction rules of the modeling language. A model written in language L_i is written $\mathcal{M}_{L_i}$. The meaning of $\mathcal{M}_{L_i}$ is established through the inter-subjectively agreed syntax and semantics of L_i.
  For each participant, the part of the externalized model which is considered relevant can be seen as a projection of the total externalized model, hence $\mathcal{M}$ can be divided into projections $\mathcal{M}^1, ..., \mathcal{M}^k$ corresponding to the participants $A_1, ..., A_k$. Generally, these projections will not be disjoint, but the union of the projections should cover $\mathcal{M}$. $\mathcal{M}$ will obviously evolve during modeling as statements are inserted and deleted.

- $\mathcal{D}$, the modeling domain, i.e. the set of all statements which can be stated about the situation at hand. If one use an objectivistic ontology, or one accept a high degree of intersubjective agreement on the modeling domain, this is similar to the definition of the original framework. During devtenance of a CIS, several different although interrelated modeling domains, with accompanying models are recognised based on Iivari [139]:
  - The existing IS as it is perceived, $\mathcal{M}(EIS)$. Another description of this is the internalisation of the current organizational reality.
  - A future IS as it is perceived, i.e requirements to a future IS, $\mathcal{M}(FIS)$.
  - The future CIS as it is perceived, i.e. requirements to a future CIS, $\mathcal{M}(FCIS)$.
  - The implemented CIS. As discussed earlier, also the CIS can be regarded as a model, although usually not a conceptual model in the sense we use the term. To be able to discuss traceability and traditional quality aspects, we also include this. The use of $\mathcal{M}(CIS)$ is also useful for controling the evolution of the CIS, and can become part of $\mathcal{M}(EIS)$.

  The domain evolve during modeling, both because of the modeling and because of external changes.
- $\mathcal{K}$, the relevant explicit knowledge of the audience, i.e. the union of the set of statements, $\mathcal{K}_1$,...,$\mathcal{K}_k$, one for each participant. $\mathcal{K}_i$ is all possible statements that would be correct and relevant for addressing the problem at hand according to the explicit knowledge of participant A_i. $\mathcal{K}_i \subset \mathcal{K}^i$, the explicit internal reality of the social actor A_i. $\mathcal{M}_i$ is an externalisation of $\mathcal{K}_i$ and is a model made on the basis of the knowledge of the individual or organizational actor. Even if the internal reality of each individual will always differ, the explicit internal reality concerning a constrained area might be equal, especially within certain groups of participants [102, 248], thus it can be meaningful to also speak about the explicit knowledge of an organizational actor. $\mathcal{M}_i \setminus \mathcal{M}^i = \emptyset$, whereas the opposite might not be true, i.e. more of the total externalized model than the part which is an externalisation of parts of an actors internal reality is potentially relevant for this actor. $\mathcal{K}$ will and should change during modeling to achieve both personal and organizational learning [317].
- $\mathcal{I}$, the audience interpretation, i.e. the set of all statements which the audience perceive that an externalized model consists of. Just like for the externalized model itself, its interpretation can be projected into $\mathcal{I}_1, ..., \mathcal{I}_n$ denoting the statements in the externalized model as they are perceived by each social actor. In addition can the model also be projected into $\mathcal{I}_{n+1}, ..., \mathcal{I}_m$ denoting the statements in the conceptual model as they are interpreted by each technical actor in the audience.

The primary goal for semantic quality is a correspondence between the externalized model and the domain as before, but this correspondence can neither be established nor checked directly: to build the model, one has to go through the participants' knowledge regarding the domain, and to check the model one has to compare this with the participants' interpretation of the externalized model. Hence, what we observe at quality control is not the actual semantic quality of the model, but a *perceived semantic quality* based on comparisons of the two imperfect interpretations.

Table 6.1 shows an overview of the goals and means as has been identified on the different semiotic levels. Language quality goals are looked upon as means in the framework. Means being goals or means on "lower" levels in the semiotic ladder are not restated when they reappear on the higher level. We will discuss each aspect in more detail below after giving a more detailed definition of the term 'statement' which are used extensively in the framework.

Quality types	*Goals*	*Means*	
		Model and language properties	Activities
Physical quality	Externalisation	Domain appropriateness	
		Participant knowledge appropriateness	
	Internalizability	Persistence	DB-activities
		Availability	
Syntactic quality	Syntactic correctness	Formal syntax	Error prevention
			Error detection
			Error correction
Semantic quality	Feasible validity	Formal semantics	Consistency checking
	Feasible completeness	Modifiability	Statement insertion
			Statement deletion
			Driving questions
Perceived sem.quality	Perceived validity	Variety	Participant training
	Perceived completeness		
Pragmatic quality	Feasible comprehension	Expressive economy	Inspection
		Aesthetics	Visualisation
			Filtering
			Diagram layout
			Rephrasing
			Paraphrasing
			Explanation
		Executability	Execution
			Animation
			Simulation
Social quality	Feasible agreement	Inconsistency handling	Model integration
			Conflict resolution
Knowledge quality	Feasible knowledge completeness		Stakeholder identification
	Feasible knowledge validity		Participant selection

Table 6.1: Framework for model quality

6.2.1 The Term 'Statement'

Statement : A *sentence* representing one *property* of a certain *phenomenon.*

To be able to count statements in different modeling languages, one needs to have specified a canonical form of the language, into which specific model-instances can be translated. The canonical form should be possible to translate into binary predicates as illustrated below.

More concretely, in the example below we have used the ER-model from Chapter 5.2.1 to illustrate in principle which statements it can be said to includes. The binary form used here is similar to the one used in Appendix E. It is important to be aware of that this is only one of many possible encodings of this form.

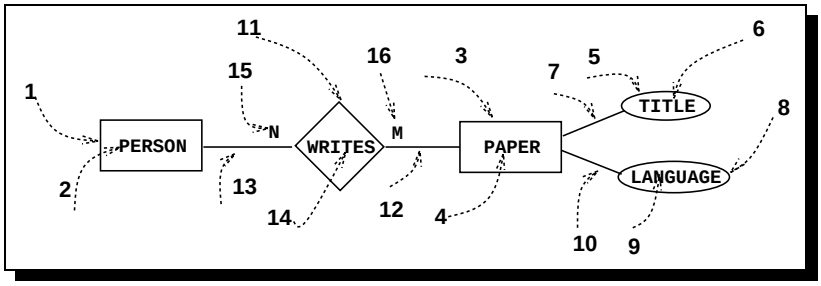

Figure 6.4: Statements in a simple ER-diagram

- 1. There exist an entity-class. EXIST(PHENOMENA, E1), PHENOMENATYPE(E1,STATIC)
- 2. The label of the entity-class E1 is 'person'. REFERENCE(E1,'PERSON')
- 3. There exist another entity-class. EXIST(PHENOMENA, E2), PHENOMENATYPE(E2,STATIC)
- 4. The label of the entity-class E2 is 'paper'. REFERENCE(E2,'PAPER')
- 5. There exist a type. EXIST(PHENOMENA, T1), PHENOMENATYPE(T1,STATIC)
- 6. The label of type T1 is 'title'. REFERENCE(P1,'TITLE')
- 7. One property of paper is title. EXIST(PHENOMENA,A1), PHENOMENATYPE(A1,RULE), DOMAIN(A1,E1), RANGE(A1,T1)
- 8. There exist another type. EXIST(PHENOMENA, T2), PHENOMENATYPE(T2,STATIC)
- 9.The label of type T2 is 'language'. REFERENCE(T2,'LANGUAGE')
- 10. One property of paper is language EXIST(PHENOMENA,A2), PHENOMENATYPE(A2,RULE), DOMAIN(A2,E1), RANGE(A2,T2)
- 11. There exist a relationship. EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,STATIC)
- 12. Relationship R1 include person. AGGREGATION(R1,E1)
- 13. Relationship R1 also include paper. AGGREGATION(R1,E2)
- 14. The label of relationship R1 is 'writes'. REFERENCE(R1,'WRITES')
- 15. One paper can be written by more than one person. EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,RULE), MODALITY(R1, PERMITTED), PRECONDITION(PERSON.X WRITES[PAPER.Y] POSTCONDITION(PERSON.Z WRITES[PAPER.Y] Z$\neq$X)
- 16. One person can write more than one paper EXIST(PHENOMENA,R2), PHENOMENATYPE(R2,RULE), MODALITY(R2, PERMITTED), PRECONDITION(PERSON.X WRITES[PAPER.Y]), POSTCONDITION(PERSON.X WRITES[PAPER.Z] Z$\neq$Y)

The above amounts to a total of 34 statements. Similarly can be defined for additional conceptual modeling languages as illustrated in Appendix E. The use of a general language-model is useful for supporting this in the general case, but we will not go into details about this here.

6.2.2 Physical Quality

Although information system models are not usually of the physical kind, any model can be represented physically somehow, e.g. on disk or paper. In our example, it is basically represented on paper as Figure 5.1. The basic quality features on the physical level is externalisation, that the knowledge of some social actor has been externalized by the use of a conceptual modeling language, and internalizability, that the externalized model is persistent and available enabling the audience to make sense of it. This is not the same as internalization of the model. Actual internalisation of the model is looked upon after the discussion of pragmatic and social quality below.

Externalisation can be defined as:

$$externalization = 1 - \frac{\#(\mathcal{K} \backslash \mathcal{M})}{\#(\mathcal{K})} \tag{6.1}$$

The major means for achieving this are the domain and participant knowledge appropriateness of the modeling language used, as will be discussed under language quality below.

Internalizability on the physical level has two primary means, persistence and availability:

- Persistence: How persistent is the model, how protected is it towards loss or damage? This also includes previous versions of the model, if these are relevant. A previous version of the model will be part of the modeling domain and it might be necessary to model the relationship to this.
- Availability: How available is the model to the audience? Clearly, this is dependent on its externalization and since the model is likely to be evolving, availability also depends on distributability, especially if members of the audience are geographically dispersed. Then, a model which is in an electronically distributable format will be more easily propagated than one which must be printed on paper and sent by ordinary mail or fax. It may also matter exactly what is distributed, e.g. the model in an editable form or merely in an output format.

Some of the activities in connection with physical quality are typical based on traditional database-functionality such non-redundant storage of data, high level access, data independence, querying and reporting facilities, real-time update, locking, concurrency, integrity, security, and recovery [132]. In addition it is regarded necessary for advanced tools for conceptual modeling and system development to include functionality such as version control and configuration management and advanced concurrency control mechanism, that are not normally found in conventional DBMSs [132].

6.2.3 Syntactic Quality

Syntactic quality is the correspondence between the model $\mathcal{M}$ and the language extension $\mathcal{L}$ of the language in which the model is written. There is only one syntactic goal, **syntactical correctness**, meaning that all statements in the model are according to the syntax and vocabulary of the language i.e.

$$\mathcal{M}_E \setminus \mathcal{L} = \emptyset \tag{6.2}$$

The degree of syntactic quality can be measured as one minus the rate of erroneous statements,

i.e.

$$syntactic\,quality = 1 - \frac{\#(\mathcal{M}_E \setminus \mathcal{L})}{\#\mathcal{M}_E} \tag{6.3}$$

Syntax errors are of two kinds:

- **Syntactic invalidity**, in which words or graphemes not part of the language are used. An example is given in Figure 6.5, where an actor-symbol not being part of the language is introduced.

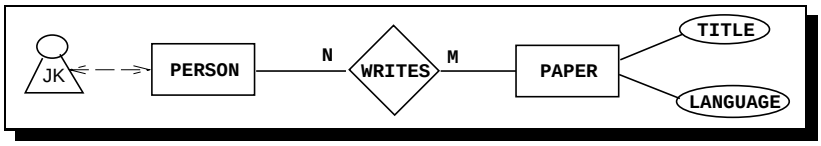

Figure 6.5: Example of syntactic invalidity

- **Syntactic incompleteness**, in which the model lacks constructs or information to obey the language's grammar. An example is given in Figure 6.6, where only one of the entity-classes that take part in the relationship is indicated.

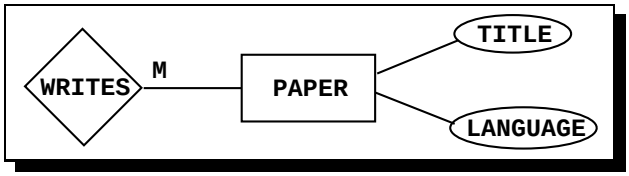

Figure 6.6: Example of syntactic incompleteness

There are three basic syntactic means: error prevention, error detection, and error correction. All are easier if the languages used have a formal syntax. Prevention is to reject the insertion of syntactically incorrect statements in the model. Detection is detecting errors after the statements have been inserted. Prevention is most useful for syntactic invalidity using a syntax directed editor, detection for syntactical completeness. Error correction - to replace a detected error with a correct statement - is more difficult to automate.

6.2.4 Semantic Quality

Semantic quality is the correspondence between the model and the modeling domain [199].

The framework contains two semantic goals; validity and completeness.

- **Validity** means that all statements made in the model are regarded as correct and relevant to the problem, i.e.

$$\mathcal{M} \setminus \mathcal{D} = \emptyset \tag{6.4}$$

A definition for the degree of validity could be

$$validity = 1 - \frac{\#(\mathcal{M}_E \setminus \mathcal{D})}{\#\mathcal{M}_E} \tag{6.5}$$

however, it can be questioned how useful such a metric might be, since it can never be measured due to the intractability of the domain. An example of invalidity is given in Figure 6.7 where the attribute 'maximum speed' is added to the entity 'paper' something we believe most persons would agree to be invalid.

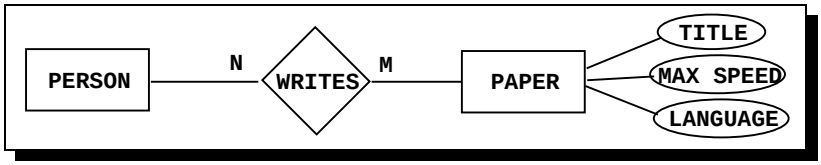

Figure 6.7: Example of semantic invalidity

- **Completeness** means that the model contains all the statements which would be correct and relevant about the domain, i.e.

$$\mathcal{D} \setminus \mathcal{M} = \emptyset \tag{6.6}$$

A definition for the degree of completeness could similarly be

$$completeness = 1 - \frac{\#(\mathcal{D} \setminus \mathcal{M})}{\#\mathcal{D}} \tag{6.7}$$

This would only be interesting in limited domains, say e.g. that it is temporarily decided upon a model of a new CIS. Then one would like to see all the statements in the model also being part of the implemented CIS. On the other hand, $\mathcal{D}$ is not completely held in the previous model, thus validity is also in this case more relevant. An example of incompleteness can be the original Figure 5.1, missing 'name' as an attribute of 'person', something we believe most persons would regard as important to represent in connection to a conference system.

For anything but extremely simple and highly inter-subjectively agreed domains, total validity and completeness cannot be achieved. Hence, for the semantic goals to be realistic, they have to be somewhat relaxed, by introducing the idea of *feasibility*. Attempts at reaching a state of total validity and completeness will lead to unlimited spending of time and money on the modeling activity. The time to terminate a modeling activity is thus not when the model is "perfect" (which will never happen), but when it has reached a state where further modeling is regarded to be less beneficial than applying the model in its current state. With respect to this, a relaxed kind of validity and completeness can be defined. To represent this we first define the insertion of a statement into a model, and the deletion of a statement from a model.

Inserting a statement s in model $\mathcal{M}$ is a mapping from the existing model, into another model that also include the inserted statement.

$$I_{M,s} = \mathcal{M} \cup \{s\} \tag{6.8}$$

Similarly can be defined for the insertion of $\mathcal{S}$, a set of statements.

A *syntactically complete* insert would in addition ensure that the model after the insertion is syntactically complete cf. Section 6.2.3.

Deleting a statement s from model $\mathcal{M}$ is a mapping from the existing model into a model where the specified statement is no longer included.

$$D_{M,s} = \mathcal{M} \setminus \{s\} \tag{6.9}$$

Similarly can be defined for the deletion of $\mathcal{S}$, a set of statements.

A *syntactically complete* delete would in addition ensure that the model after the deletion is syntactically complete cf. Section 6.2.3. This can be supported in a modeling tool using a similar technique as the filtering-technique presented in [282], having a neighbor function $\mathcal{N}$ for each modeling symbol indicating the symbols dependent and independent of this. E.g. if one remove the statements labeled 11 in Figure 6.4 one also remove the statements labeled 12-16 as illustrated in Figure 6.8, whereas removing the statements labeled 3 also would mean to remove all other statements except the statements labeled 1 and 2. On the other hand, this might not always be what is called for.

A syntactically complete delete can thus be defined in the following manner.

$$SCD_{M,s} = \mathcal{M} \setminus \mathcal{N}(s). \tag{6.10}$$

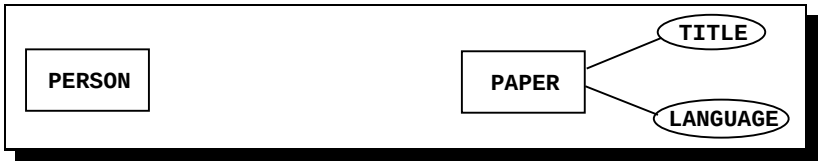

Figure 6.8: Example of a syntactically complete delete

We are now ready to define feasible validity and completeness.

- **Feasible validity:** $\mathcal{M} \setminus \mathcal{D} = \mathcal{R} \neq \emptyset$, but there is no statement $r \in \mathcal{R}$ such that the benefit of performing a syntactically valid delete of *r* from $\mathcal{M}$ exceeds the drawback eliminating the invalidity *r*.

$$(\neg\exists r \in \mathcal{R}), benefit(SCD_{M,r}) > drawback(SCD_{M,r}). \tag{6.11}$$

- **Feasible completeness:** $\mathcal{D} \setminus \mathcal{M} = \mathcal{S} \neq \emptyset$, but there is no statement $s \in \mathcal{S}$ such that the benefit of inserting *s* in $\mathcal{M}$ in a syntactically complete way exceeds the drawback of adding the statement *s*.

$$(\neg\exists s \in \mathcal{S}), benefit(I_{M,s}) > drawback(I_{M,s}). \tag{6.12}$$

Feasibility thus introduces a trade-off between the *benefits* and *drawbacks* for achieving a given model quality. We have used the term "drawback" here instead of the more usual "cost" to indicate that the discussion is not necessarily restricted to purely economical issues. Judging completeness with respect to some inter-subjectively agreed standard as suggested by Pohl [261] is one approach to feasibility.

It has earlier been shown that other suggested semantic goals like correctness, minimality, annotated, consistent, and unambiguous as used in e.g. [63] are subsumed by validity and completeness as defined here [199]. We will briefly repeat this discussion below. Note that these terms usually have been used based on an objectivistic world-view, thus when comparing them with validity and completeness as we have defined them, we do this under the presumption that the modeling domain is inter-subjectively agreed among the audience. They have also usually been used in connection with requirements specifications, and not for conceptual models.

- Correctness corresponds to validity.

- Minimality is subsumed by validity, because if $\mathcal{M}$ contains a statement that over-constrains the system, this can be regarded as an invalid statement.
- Annotated: This is subsumed by completeness. The term 'annotation' is used for associating priorities to statements and their perceived stability, which are often important aspects of the modeling domain.
- Traceability means that the origin of a statement should be clear. Since the model itself becomes part of the domain of the models that need to trace to them, the lack of important data about the model is a case of incompleteness. Also the assumptions behind the stated requirements to a new system are covered by this.
- Consistency. A model is inconsistent if it contains contradictory statements. This is subsumed by the combination of validity and incompleteness since an inconsistency must be caused by at least one invalid statement or the lack of a statement that are to sort out the inconsistency. Note that this is not necessarily the case when containing the view of several persons in the same model when the modeling domain is not inter-subjectively agreed, but this situation is neither discussed by other researchers that propose consistency as a goal. Note that in such cases, variety in the model is important, both to avoid model monopoly [34] and to increase the reusability of the models.
- Unambiguity: Like consistency, this is subsumed by validity and completeness. If $\mathcal{M}$ is consistent and valid, nothing is wrong with having ambiguity, except that you should state explicitly that all alternative interpretations are intended. Without this explicit statement, there is incompleteness.

Activities for establishing higher semantic quality, are statement insertion and deletion. An update is a deletion followed by an insertion. Statement insertions and deletions can obviously also result in lower semantic quality. Statement insertion and deletion can generally be looked upon as meaning updating transformations, which can be done either manually or automatically. Examples of the latter is for instance the situation with a syntactically complete delete as discussed above. Another example is the evolution transformations described in [151] which is an example of idea reuse.

Consistency checking is another activity here. To be able to do consistency checking, the model must be made in a formal, preferably logical language, and to enable and assess the impact of updates, it should be modifiable. This includes properties such as structure, locality of changes, and control of redundancy. The final activity we describe is the use of driving questions based on the already existing model as used in Tempora [324]. For feasibility analysis, however, there is little in way of tool support, except when the domain is already described in another model, or a standard has been agreed upon. General tools are difficult to develop for the simple reason that the domain and audience are beyond automatic manipulation. We will return to this in Chapter 10.

6.2.5 Perceived Semantic Quality

Perceived semantic quality is the correspondence between the actor interpretation of a model and his or hers current knowledge of the domain. Perceived validity and completeness can be expressed as indicated below:

- *Perceived validity* of the model externalization: $\mathcal{I}_i \setminus \mathcal{K}_i = \emptyset$.
- *Perceived completeness* of the model externalization: $\mathcal{K}_i \setminus \mathcal{I}_i = \emptyset$.

Metrics for the degree of perceived validity and completeness can be defined by means of cardinalities the same ways as for syntactic quality.

$$perceived\ validity = 1 - \frac{\#(\mathcal{I}_i \setminus \mathcal{K}_i)}{\#(\mathcal{I}_i)} \tag{6.13}$$

,i.e. the number of invalid statements interpreted, divided by the total number of statements interpreted by the actor A_i. An example on a model with a perceived invalid statement is the example in Figure 5.1, where I, in the role of an end-user of a conference system, would not regard the 'language' attribute to be relevant for 'paper'.

$$perceived\ completeness = 1 - \frac{\#(\mathcal{K}_i \setminus \mathcal{I}_i)}{\#(\mathcal{K}_i)} \tag{6.14}$$

, i.e. the number of relevant statements known but not seen in the model, divided by the total number of relevant knowledge statements known by the actor A_i. Also on these measures, a discussion of feasibility is useful. As on semantic quality, I miss among other things the name of person in the model in Figure 5.1.

The perceived semantic quality of the model can change in many ways:

- A statement is added to $\mathcal{M}^i$ which is understood to be in accordance to the knowledge of actor A_i, thus increasing perceived completeness.
- A statement is added to $\mathcal{M}^i$ which is understood to not be in accordance to the knowledge of actor A_i, thus decreasing perceived validity.
- A statement is removed from $\mathcal{M}^i$ that earlier was understood not to be in accordance with the knowledge of actor A_i, thus increasing perceived validity.
- A statement is removed from $\mathcal{M}^i$ that earlier was understood to be in accordance with the knowledge of actor A_i, thus decreasing perceived completeness.
- $\mathcal{K}_i$ changes, which can both increase and decrease perceived validity and completeness of the model. One way $\mathcal{K}_i$ can change, is through the internalization of another model made on the basis of the knowledge of another actor. Internalization will be discussed further after discussing social quality.
- The actor's knowledge of the modeling language changes, potentially changing $\mathcal{I}_i$ which can both increase and decrease the perceived validity and completeness of the model.

The means for achieving a high perceived validity and completeness are similar to the ones for traditional validity and completeness, with the addition of participant training.

6.2.6 Pragmatic Quality

Pragmatic quality is the correspondence between the model and the audience's interpretation of it. The framework contains one pragmatic goal, namely **comprehension**. Not even the most brilliant solution to a problem would be of any use if nobody was able to understand it. Moreover, it is not only important that the model has been understood, but also *who* has understood it.

Individual comprehension is defined as the goal that the individual actor A_i understands the part of the model relevant to that actor, i.e. $\mathcal{I}_i = \mathcal{M}^i$.

For total comprehension, one must have

$$(\forall i, i \in [1 \ldots k])(\mathcal{I}_i = \mathcal{M}^i) \tag{6.15}$$

, i.e., that every individual actor understands the part of $\mathcal{M}$ relevant for him/her.

The corresponding error class is *incomprehension*, meaning that the above formula does not hold.

For a large model, it is unrealistic to assume that each audience member will be able to comprehend the consequences of all the statements which are relevant to them. Thus, comprehension as defined above is an ideal goal, just like validity and completeness, and can often not be achieved. Again it will be useful to introduce the notion of feasibility:

Feasible comprehension means that although the model may not have been correctly understood by all audience members, i.e.

$$(\exists i)(\mathcal{I}_i \setminus \mathcal{M}^i) \cup (\mathcal{M}^i \setminus \mathcal{I}_i) = \mathcal{S}_i \neq \emptyset, \tag{6.16}$$

there is no statement $s \in \mathcal{S}_i$ such that the benefit of rooting out the misunderstanding corresponding to *s* exceeds the drawback of taking that effort.

It is important to notice that the pragmatic goal is stated as *comprehension*, i.e. that the model has been understood, not as *comprehensibility*, i.e. the model's ability to be understood. There are several reasons for doing so. First, the ultimate goal is that the model is understood, not that it is understandable. Moreover, it is hard to speak about the comprehensibility of a model as such, since this is so much dependent on the process by which it is developed, the way the participants communicates with each other and various kinds of tool support.

From the technical actors' point of view, that a model is understood means that $(\forall i, i \in [n+1...m])\mathcal{I}_i = \mathcal{M}^i$, i.e. all statements that are relevant to the technical actor to be able to perform code generation, simulation, etc. are comprehended by this actor. In this sense, formality can be looked upon as being a pragmatic goal, formal syntax and formal semantics are means for achieving pragmatic quality. This illustrates that pragmatic quality is dependent on the different actors. This also applies to social actors. Whereas some individuals from the outset are familiar with formal languages, and a formal model in fact will be best for them also for comprehension, other individuals will find a mix of formal and informal statements to be more comprehensive, even if the set of statements in the complete model is redundant.

Some of the activities to achieve pragmatic quality are:

- Audience training: Educate the audience in the syntax and semantics of the modeling languages.
- Inspection: Manually reading a model.
- Transformations: Generally to transform a model into another model in the same language. This can generally be expressed as

$$T : \mathcal{M}1_{L_i} \rightarrow \mathcal{M}2_{L_i} \tag{6.17}$$

Rephrasing is a meaning preserving transformation where some of the implicit statements of the model is made explicit. One example is the use in ERAE [119] applying logical rules such as

$$\phi \rightarrow \psi \Rightarrow \neg\psi \rightarrow \neg\phi$$

Layout modification is another kind of meaning-preserving transformation which can improve the comprehension of the models. A layout modification is described as a mapping from a set of statement into the same set of statements, although they might be differently arranged spatially.

$$L : \mathcal{M} \rightarrow \mathcal{M} \tag{6.18}$$

i.e. a pure layout modification do not change the meaning of the model.
A list of guidelines for graph aesthetics is presented in [302], reproduced in Table 6.2, and this could be a possible starting point for automatic layout modification. Another use can

be to produce metrics, e.g. the number of crossing lines divided by the number of links in total in a figure, or compared with the minimum possible number of crossing as long as one do not duplicate symbols. Similar metrics can be devised for the other aesthetics, and be used during modeling to assess the potential for improving aesthetics. Based on such metrics one could assess that the quality of Figure 6.9 is less than that of Figure 5.1 on this account although it contains the same statements. On the other hand, we should remember that aesthetics is a subjective issue, thus familiarity with a diagram is often just as important for comprehension. As noted by [257] one of the main advantages of diagrammatic modeling languages appears to be the use of so-called secondary notation, i.e. the use of layout and perceptual cues to improve comprehension of the model. Thus, one need often to constrain automatic layout modifications. A list of constraints used in connection to this is given in Table 6.3.

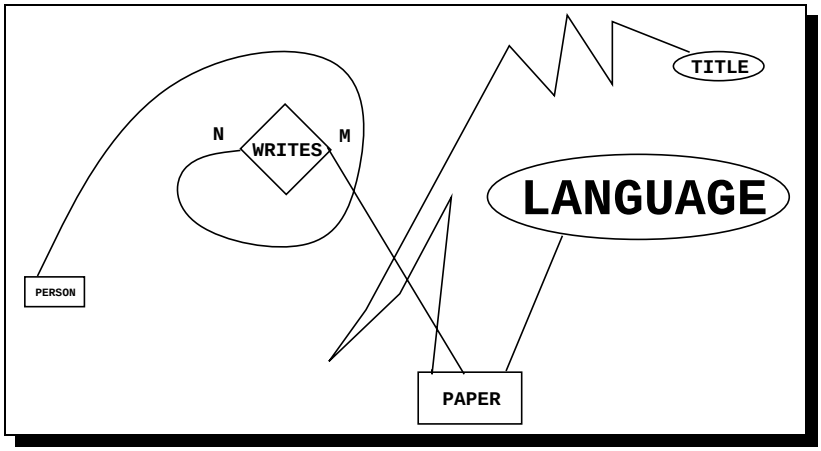

Figure 6.9: Example on poor aesthetics

Referring to the semiotic ladder in FRISCO, some might claim that such aesthetics should rather have been listed as goals for empirical quality. However, aesthetics are related to humans' possibility to comprehend, and are thus most conveniently presented as a pragmatic mean in our framework to emphasize the link to comprehension.

ANGLE	Angles between edges should not be too small.
AREA	Minimize the area occupied by the drawing.
BALANCE	Balance the diagram with respect to the axis.
BENDS	Minimize the number of bends along the edges.
CONVEX	Maximize the number of faces drawn as convex polygons.
CROSSING	Minimize the number of crossings between edges.
DEGREE	Place nodes with high degree in the center of the drawing.
DIM	Minimize differences among nodes' dimensions.
LENGTH	Minimize the global length of edges.
MAXCON	Minimize length of the longest edge.
SYMMETRY	Have symmetry of sons in hierarchies.
UNIDEN	Have uniform density of nodes in the drawing.
VERT	Have verticality of hierarchical structures.

Table 6.2: A taxonomy of graph aesthetics (From [302])

Filtering is a meaning removing transformation, concentrating on and illuminating specific parts of a model. Filtering has been defined in [282] based on the notion of a *viewspec*

CENTER	Place a set of given nodes in the center of the drawing.
DIMENSION	Assign the dimensions of symbols.
EXTERNAL	Place specified nodes on the external boundary of the drawing.
NEIGH	Place a group of nodes close.
SHAPE	Draw a subgraph with a predefined shape.
STREAM	Place a sequences of nodes along a straight line.

Table 6.3: A taxonomy of constraints for graph layout (From [302])

$\mathcal{V}$, which is a model containing a subset of the statements of another model in the same language i.e. $\mathcal{V} \subseteq \mathcal{M}$.
Filters can be classified into two major groups:

- Language/meta-model filters: Suppress details with respect to graphemes and symbols in the modeling language. An example is illustrated in Figure 6.10 where all attributes are removed.

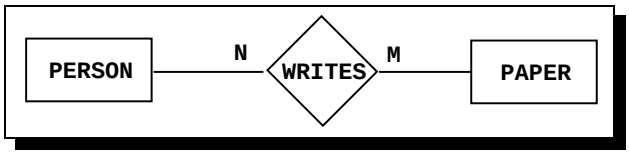

Figure 6.10: Example of a language filter

- Model/specification filters: Suppress details with respect to a particular model. An example is given in Figure 6.11 where only the attributes and subclasses of a selected entity-class, in this case 'paper', are retained.

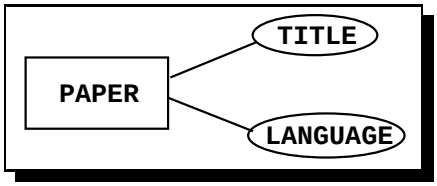

Figure 6.11: Example of a model filter

- Translation: A translation can generally be described as a mapping from a model in one language to a model containing all or some of the same statements in another language:

$$T : \mathcal{M}_{L_i} \rightarrow \mathcal{M}_{L_j}, i \neq j \tag{6.19}$$

In *paraphrasing* both L_i and L_j are textual languages.
In *visualization*, L_i is a textual language whereas L_j is a diagrammatical language.
Translations between different diagrammatical languages can also useful for comprehension in the case different persons are fluent in different related languages. For example for those being familiar with GSM, the diagram in Figure 6.12 might be better for comprehension than the diagram in Figure 5.1.
Finally, one might want to translate a diagrammatical model into a textual language, for instance a programming language so that the resulting model can be executed. In this case L_i is diagrammatical and L_j is textual.

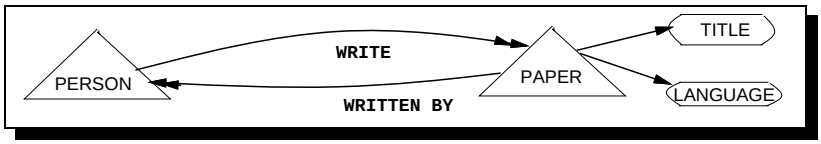

Figure 6.12: The example written in the GSM language

The extent of meaning preservation in the translations depends on both languages in question and how they relate to each other.

A *complete translation* is a translation where all statements in the source model are also contained in the target model. i.e. the mapping is an injection.

A *valid translation* is a translation in which all statements in the target model are also contained in the source model, i.e. the mapping is a surjection. For a translation that is both complete and valid, we have a bijection. It can also be useful to distinguish a completely traceable translation, where all statements in the target model is based on and can be traced back to the source model.

Most translations and transformations will be easier and faster to perform when having tool support, but they can also be done manually. Manual translations and transformations can also be used as part of participant training [98]. On the other hand, also audience training might be enhanced by using tool support. Several specific applications of translations and transformations and combinations of these exist:

- Model execution: Translate the model to a model in an executable language, e.g. the languages used for the resulting CIS, and execute this model [125, 335].
- Animation: Make systems dynamics explicit by using moving pictures. This might take the form of icons such as a telephone ringing or a customer arriving at a registration desk, or it might apply the symbols of the modeling language. Animation is practically impossible without tool support.
- Explanation generation: This can be manual or tool-supported. An explanation generator can answer questions about a model and its behavior [114].
- Simulation: Use statistical assumptions about the domain such as arrival rate of customers and distributions of processing times, to anticipate how a system built according to the model would behave if implemented. Neither this is practical without tool support for large models. Simulation can be combined with execution, animation, and explanation [125].

The properties a model and the languages it is made in must have include those for syntactic and semantic quality, as well as executability (i.e. the execution of the model has to be efficient), expressive economy, and aesthetics as mentioned above.

6.2.7 Social Quality

The goal defined for social quality is *agreement*. Four kinds of agreement can be identified, according to binary distinctions along two orthogonal dimensions:

- Agreement in knowledge vs. agreement in model interpretation.
- Relative agreement vs. absolute agreement.

Relative agreement means that the various projections are consistent — hence, there may be many statements in the projection of one actor that are not present in that of another, as long as they do not contradict each other. Absolute agreement, on the other hand, means that all projections are the same.

Agreement in model interpretation will usually be a more limited demand than agreement in knowledge, since the former one means that the actors agree about what is stated in the model, whereas there may still be much they disagree about which is not stated in the model so far, even if it might be regarded as relevant by one or both participants.

Hence, we can define

- Relative agreement in interpretation: all I_i are consistent,
- Absolute agreement in interpretation: all I_i are equal,
- Relative agreement in knowledge: all K_i are consistent,
- Absolute agreement in knowledge: all K_i are equal,

and metrics can be defined for the degree of agreement based on the number of inconsistent statements divided by the total number of statements perceived, or by the number of uncorresponding statements divided by the total number of statements perceived. The equation below specify a metric for relative agreement in interpretation(RAI).

$$RAI = 1 - \frac{\#(\{s \mid (\exists i, j, i \neq j), s \in \mathcal{I}_i \wedge \neg s \in \mathcal{I}_j\})}{\#(\mathcal{M}_E)} \tag{6.20}$$

Since different participants will have their expertise in different fields, relative agreement is regarded to be more useful than absolute agreement. On the other hand, the different actors must have the *possibility* to agree and disagree on something, i.e. the parts of the model which are relevant to them should overlap to some extent. This can be expressed as $(\forall i)\mathcal{M}^i \cap (\mathcal{M}^1 \cup \cdots \cup \mathcal{M}^{i-1} \cup \mathcal{M}^{i+1} \cup \cdots \cup \mathcal{M}^k) \neq \emptyset$

It is not given that all participants will come to agreement. Few decisions are taken in society under complete agreement, and those that are are not necessarily good, due to group-think and other detrimental factors. To answer this we introduce *feasible agreement*:

Feasible agreement is achieved if feasible perceived semantic quality and feasible comprehension are achieved and inconsistencies between statements in the different interpretations of the model $\mathcal{I}_i$ are resolved by choosing one of the alternatives when the benefits of doing this is less than the drawbacks of working out an agreement.

The pragmatic goal of comprehension is looked upon as a social mean. This because agreement without comprehension is not very useful, at least not when having democratic ideals. Obviously if someone is trying to manipulate a situation, agreement without comprehension is useful.

Some activities for achieving feasible agreement are model integration and conflict resolution. The general process has many similarities with view integration, which has been a topic of much research in the database community. The process can be considered as consisting of four subprocesses [94].

- Pre-integration: When more than two models are used as input to the process, one must decide on how many models should be considered at a time. A number of strategies have been developed such as [83]: binary ladder integration, N-ary integration, balanced binary strategy, and mixed strategies.
- Viewpoint comparison: Includes identifying correspondences and detecting conflicts among the viewpoints.
- Viewpoint conforming: Aims at solving the previously detected conflicts. Representations of statements in two different models can be classified as follows [94]: Identical, equivalent, compatible, and inconsistent. To deal with such conflicts traditional approaches are mostly based on either transformational equivalence or they entrust the skill of the participants by providing only examples valid for the particular model. According to [94] few approaches deal with inconsistent statements. A notable exception is [190]. Other useful

techniques in this respect is the use of argumentation systems [56] for supporting the argumentation process.

- Merging and restructuring: The different models are merged into a joint model and then restructured. The latter involves checking the resulting model against criteria for semantic, pragmatic, and social quality.

Generally, it is not to be expected and is neither beneficial that the matching process apart from syntactic matching can be performed totally automatic. Model merging can be supported in several ways, having computerized support for manual integration, possibly with the use of CSCW-techniques. This is discussed in more detail in [4, 282].

6.2.8 Internalization

Internalisation of a model happens as a result of comprehension and agreement around statements not being part of the model made as an externalisation of the persons existing internal reality.

Internalisation as part of conceptual modeling can be expressed crudely as a mapping between the sets of statements, being part of the explicit internal reality of an actor.

$$\mathcal{K}_{i_{t+1}} = (\mathcal{K}_{i_t} \cup (\mathcal{N} \subset \mathcal{M}_j)) \setminus (\mathcal{O} \subset \mathcal{K}_{i_t}) \tag{6.21}$$

$i \neq j, \mathcal{O} \cap \mathcal{N} = \emptyset, \mathcal{K}_{i_t} \setminus \mathcal{N} = \mathcal{K}_{i_t}$

$\mathcal{N}$ and $\mathcal{O}$ above is sets of statements. If $\mathcal{O} = \emptyset$ this is a monotonous growth of $\mathcal{K}_i$. If $\mathcal{O} \neq \emptyset$ there is a non-monotonous growth of $\mathcal{K}_i$.

6.2.9 Knowledge Quality

From a standpoint of social constructivity, it is difficult to talk about the quality of explicit knowledge. On the other hand, within certain areas, for instance mathematics, what is generally regarded as "true" is comparatively stable, and it is inter-subjectively agreed that certain persons have more valid knowledge of an area than others. It is important to keep in mind that their knowledge is only partial. The "quality" of the participant knowledge can thus be expressed by the relationships between the audience knowledge and the domain. The "perfect" situation would be if the audience knew everything about the domain at a given time.

$$Knowledge\ completeness : \mathcal{D} \setminus \mathcal{K} = \emptyset \tag{6.22}$$

and that they had no incorrect superstitions about the domain, i.e.,

$$Knowledge\ validity : \mathcal{K} \setminus \mathcal{D} = \emptyset \tag{6.23}$$

To get a *good enough* knowledge about the domain, careful *participant selection* based on *stakeholder identification* is necessary (if you have a problem and can choose the participants), or alternatively, careful *problem selection* (if the participants are given, but not the problem to be solved). In the case that both participants and problem are more or less given, and not fitting too well, some development in terms of training of the participants or modification of the problem may be necessary. Just as for the other aspects of quality, it will be possible to talk about *feasible knowledge quality*, meaning that the knowledge of the audience could still be improved, but the benefit of improving it through additional education or the hiring of additional experts or including additional stakeholders will be less than the drawbacks of mistakes made due to imperfect

knowledge. In the view of social construction every stakeholder might have something unique to contribute to the process of conceptual modeling. On the other hand, it is obviously not feasible to include say 500 end-users and 200.000 indirect users (customers) of the organization in constructing the requirements of a new application system.

6.2.10 Language Quality

The following discussion is based on the overview originally made by Sindre [286], and extended by Seltveit [282]. Their results are rearranged to fit with the categories in the framework for model quality. The given criteria are to some extent a matter of belief and taste, but is generally based on results from linguistics and psychology.

In the original framework, distinctions were made along two dimensions. First, it was distinguishes between two main kinds of criteria:

- Criteria for the underlying basis of the language.
- Criteria for the external representation of the language.

For each of these two parts, the following four main groups of criteria were identified:

- **Perceptibility**: How easy is it for persons to comprehend the language? This is related to the current and potential knowledge of the participants and their interpretation of models in the language.
- **Expressive power**: What is it possible to express in the language? This is related to the domain.
- **Expressive economy**: How effectively can things be expressed in the language? This is also related to participant interpretation.
- **Method/tool potential**: How easily does the language lend itself to proper method and tool support? This is related to the capabilities of the technical actors in the audience.

In addition, Seltveit introduced reducibility as a separate category, meaning what features is provided by the language to deal with large and complex systems.

We have regrouped the factors according to the framework for model quality as follows:

- **Domain appropriateness**.
- **Participant knowledge appropriateness**.
- **Participant interpretation enhancement**.
- **Technical actor interpretation enhancement**.

Since participant interpretation is done on the basis of the current participant knowledge, factors 2 and 3 will be closely intertwined.

We will continue to distinguish between the underlying basis of a language and its external representation since it will result in a clearer discussion, but do this on a factor to factor basis. As in the original framework, different criteria in the different factors will often be contradictory, i.e. one should expect to find some deficiencies for most conceptual modeling languages based on goals for language quality. On the other hand, this can be addressed by how the language is used within a methodology.

Domain appropriateness

Domain appropriateness can be describes as follows:

$$\mathcal{D} \setminus \mathcal{L} = \emptyset \tag{6.24}$$

i.e. there are no statements in the domain that can not be expressed in the language. Obviously this means that different languages are more or less suitable for different problem situations. In Chapter 5 the existing perspectives used and deemed necessary were discussed, and we will refer to this discussion when evaluating this aspect later in the thesis.

- Underlying basis: Ideally, this must be powerful enough to express anything in the domain. There is an infinite number of statements that we might need to make, and these have to be dealt with through a rather small number of phenomena classes due to the participant interpretation as will be discussed below. This means that
  - The phenomena must be *general* rather than specialized.
  - The phenomena must be *composable*, which means that we can group related statements in a natural way. When only domain appropriateness is concerned, it is an advantage if all thinkable combinations are allowed, each yielding a separate meaning.
  - The language must be *flexible in precision:*
    - To express precise knowledge one need precise constructs. This means that the language must be *formal* and *unambiguous*.
    - At the same time, one need vague constructs for modeling vague knowledge. To fulfill both requirements, the vagueness must also be formalized (i.e. even the vague constructs must have a definite interpretation — the constructs are called vague because their interpretation is wide compared to the more definite constructs).
- External representation: The only requirement to the external representation is that it does not destroy the underlying basis. Thus,
  - Every possible statement in the language should have a unique representation in the basis (otherwise, the precision of the language will be destroyed).
  - Every possible statement in the underlying basis should have at least one external representation (otherwise, the generality might be destroyed).
  - Just like the phenomena, the symbols of the language must be *composable*.

  As indicated, only the mapping from symbols to phenomena needs to be unique — it is all right to have several alternative external representations for the same statement.

Participant knowledge appropriateness

This can be expressed by:

$$(\forall i \in [1...k])\forall j (\mathcal{M}_{L_j})^i (\mathcal{M}(L_j) \setminus \mathcal{K}^i) = \emptyset \tag{6.25}$$

i.e. all the statements in the language models of the languages used by the different participants are part of the explicit knowledge of this participant.

- Underlying basis: This should correspond as much as possible to the way individuals perceive reality. This will differ from person to person and between person in different group according to their previous experience [102], and thus will initially be directly dependant on the given participants in a modeling effort. When it comes to existing use of modeling languages, Senn [283] reports that the level of awareness of structured methods (i.e. using data and process modeling languages) is high among CIS-professionals - as many as 90 percent of all analysts are familiar with these methods, according to some estimates. Approximately half of the organizations in the United States have used these methods. Based on our survey, we might expect that this number is somewhat smaller in Norway.

On the other hand the knowledge of the participants is not static, i.e. it is possible to educate persons in the use of a specific language. In that case should one base the language on experience with languages for conceptual modeling, and languages that have been used successfully earlier in similar tasks. In this connection, it is interesting to look on experiments trying to find which languages or perspectives persons find most easy to learn. Few empirical studies of this kind have been performed. Vessey and Conger [318] reports on empirical investigations among novice analysts that they seemed to have much greater difficulty applying an object methodology, than a data or process methodology. Process modeling was found easier to apply than data modeling. For experienced developers the most difficult legacy to overcome before being able to use object-orientation efficiently seems to be the investment in persons whose experience and expertise are in other way of doing things. According to Kozaczynski, to become accepted, the object-oriented way of thinking must become the natural way of thinking. Now it presents a steep learning curve [166]. In Tempora [307], the experience was that whereas participants had small problems in learning to use both the process modeling languages and the main parts of the data modeling language, the formal textual rule language was difficult for people to comprehend. On the other hand the phenomena of rules is generally well-known: As stated by Twining [312], "One reason why the notion of 'rule ' is such an important one not only in law, but in fields as varied as linguistics, sociology, anthropology, education, psychology, and philosophy, is that there is hardly any aspect of human behavior that is not governed or at least guided by rules." When it comes to the communication perspective, some experience related to learning speech-acts theory as part of using tools such as the Coordinator [89] is presented in [41]. In many cases, the users found the linguistically motivated parts of the language difficult to understand and apply. In other cases, this was not regarded as a problem. When it comes to actors and roles, we believe these phenomena classes to be easy to comprehend based on their widespread use in e.g. organizational diagrams.
Another important point in this connection is that it should be possible to express inconsistencies in the language since inconsistency between how people perceive reality is a fact of life which is useful to represent so that inconsistencies can be revealed and discussed explicitly.

- External representation: The external representation of different phenomena should be *intuitive* in the sense that the symbol chosen for a particular phenomena somehow reflects this better than another symbol would have done. Also this is partly dependant on the audience, even if general guidelines might be devised.

Participant interpretation enhancement

Similar to model interpretation, one can define language interpretation, thus the set of possible statements that can be made in the language that are understood by the audience member . Ideally

$$\mathcal{L} \setminus \mathcal{I} = \emptyset \tag{6.26}$$

i.e. all the possible statements of the language is understood by the participants in the modeling effort using the language.

- Underlying basis:
  - The phenomena of the language should be easily distinguished from each other.
  - The number of phenomena should be reasonable. If this has to be uncomfortably large, they should be organized hierarchically, making it possible to approach the conceptual

framework at different levels of sophistication. This hierarchical organization should in itself be natural, cf. the participant knowledge above.
  - The use of phenomena should be *uniform* throughout the whole set of statements possible to express within the language. Using the same construct for opposite linguistic functions or different constructs for the same function depending on the context will tend to make the language confusing.
  - The language must be flexible in *the level of detail:*
    * Statements must be *easily extendible* with other statement providing more details.
    * At the same time, details must be *easily hidden*.

    This means that the language must include abstraction mechanisms.
  - Separation of concerns: It is possibly to divide the models made in the language in natural parts, to be able to support work division in the sense that the individual participants can concentrate on the areas they are interested in.

  Whereas the domain appropriateness concerns what we are able to express, *expressive economy* concerns how briefly things can be expressed, i.e. how many constructs you need to make the statements you want to make. Introducing one construct for each possible statement would make every statement brief, but the number of constructs would be infinite. Since it is necessary to keep the number of construct at a reasonable level, a good expressive economy cannot be based on defining new constructs for everything. Instead
  - The most *frequent* kinds of statements should be as brief as possible.
  - The most *important* kinds of statements should be as brief as possible.
- External representation:
  - Symbol discrimination should be easy. This means that it should be easy to see the difference between the various symbols of the language.
  - The external language should be as consistent as possible, in the sense that symbol use should be uniform, i.e. a symbol should not stand for one phenomenon in one context and a completely different on in another. Neither should one use different symbols for the same phenomenon in different contexts.
  - One should strive for symbolic simplicity — both concerning the primitive symbols of the language and the way they are supposed to be connected. If the symbols themselves are visually complex, models containing a lot of symbols will be even more complex, and thus difficult to comprehend.
  - The use of emphasis in the external language should be in accordance with the relative importance of the statements. Factors that have an important impact on visual emphasis are the following:
    * Size (the big is more easily noticed than the small). On the other hand, nodes in graphs should not have very different dimensions.
    * Solidity (e.g. **bold** letters vs. ordinary letters, full lines vs. dotted lines, thick lines vs. thin lines, filled boxes vs. non-filled boxes).
    * Difference from the ordinary pattern (e.g. *slanted* letters or a rare symbol will attract attention in a model of ordinary ones).
    * Foreground/background (if the background is white, things will be easier noticed the darker they are).
    * Color (red attracts the eye more than other colors).
    * Change (blinking or moving symbols attract attention).
    * Pictures vs. text (pictures usually having a much higher perceptibility, information conveyed as such will be emphasized at the cost of information conveyed

textually).
    - Degree (nodes able to connect to many others will attract attention compared to nodes making few connections).

  Areas such as size and color might also be model-specific, in case the modeling tool enables resizing etc. Emphasis is a very powerful mechanism for facilitating comprehension of models, but it can easily be overdone.
- Composition of symbols can be made in a aesthetically pleasing way, such that one can enable the creation of models with few crossing edges and short edges. The possibility of model redundancy can also be important in this respect.
- Navigation: Does the external constructs allow for nice ways of filtering, i.e. making various selections concerning which statements to show and which to hide, and browsing, i.e. moving between related symbols in a model?
- Grouping of related statements: Do the language have constructs to support the grouping of related statements?

Graphical styles of representation have some important advantages over text and tables when it comes to enhancing participant interpretation based on the possible use of secondary notation.

At the external level, expressive economy is concerned with how many symbols one need to use to express the statements of the model. As the requirements of the previous item suggest, it will usually be the case that the things easily expressed conceptually will also be easily expressed externally, and the things which are complicated in the underlying basis will also have to be complicated externally.

However, the basis and the external representation of a language should not necessarily be the same. A good external representation should always have an expressive economy better than that of the basis. This is because the external representation has many possibilities that the underlying basis does not have:

- Omission of symbols that are understood in the context.
- Special symbols can be defined for constructs which are frequent (or important).
- Multiple mentioning of the same phenomena is unavoidable at the basis level. At the external level, such multiple mentioning can often be avoided.

Of course, there are some pitfalls to be avoided.

- Blank symbols, i.e. symbols that do not contain any information for anyone.
- External redundancy, i.e. showing the same phenomena in several different ways in the same external representation.

Diagrams have a significantly larger potential for expressive economy than tables or text. On the other hand it is impossible to convey everything diagrammatically. Thus, the best thing to do for expressive economy is to try to express the frequent and most important statements diagrammatically and the less frequent textually.

Technical actor interpretation

- Underlying basis: For the technical actors, it is especially important that the language lend itself to automatic reasoning. This requires formality (i.e. both formal syntax and semantics being both operational and/or logical), but formality is not necessarily enough, since the reasoning must also be fairly efficient to be of practical use. This is covered by executability discussed under pragmatic quality.

Looking back at the discussion on pragmatic quality, formality can be defined as follows:

$$necessary\ formality = \frac{\#(\bigcup_{i=n+1}^{m} \mathcal{M}^i \cap \mathcal{L}_F)}{\#(\bigcup_{i=n+1}^{m} \mathcal{M}^i)} \tag{6.27}$$

On the other hand, a model expressed in natural language can also be useful from this point of view as techniques for natural language understanding are improved.

- Information hiding constructs: Encapsulating parts of the model limits its access from other components, i.e. to create independent parts. This is a useful property when the models are used to generate the application system, and thus simplifies testing.

6.3 Chapter Summary

We have in this chapter presented a framework for the understanding of quality of conceptual models, where model quality has been divided into physical, syntactical, semantical, pragmatical, and social aspects. Means to support the construction of high quality models have been briefly discussed, including knowledge and language quality.

We will use the proposed framework to perform evaluations of selected methodologies using a multi-perspective approach to conceptual modeling in the next chapter. What we will be able to evaluate is the *potential* for the frameworks to support the creation of conceptual models of high quality, based on the modeling techniques they support. Whereas language quality has been described in some detailed, we have only described means and activities for the creation of conceptual models of high quality on a high level. Thus the way an approach support e.g. execution of conceptual models can be more or less appropriate for achieving pragmatic quality. It is a major task to go into a detailed evaluation of how the different approaches support each activity, so we will on this area only refer to the assessments that have been brought forward in the literature regarding this. One also needs an overall methodology which suggests how do use different techniques in concert.

We will use the same framework for the evaluation of our own suggestions as they are presented in Chapter 9 and Chapter 10, where we also take a closer look at the conceptual modeling process.

Chapter 7

Multi-perspective Approaches to Conceptual Modeling

We will in this chapter look upon approaches which apply a combinations of single perspective languages as presented in Chapter 5. In this connection, we will also include some information on the methodology that has been proposed for developing conceptual models with the languages, and come with a short evaluation of the approaches.

Although the different approaches combines several perspectives, they all *emphasize* one or a few perspectives.

The following approaches are described and evaluated:

- OMT [271], being object-oriented.
- Tempora [202], being rule-oriented.
- SAMPO [14], being language-action and role-oriented.
- PPP [116], being primarily functionally oriented.

All approaches also include structural and behavioral aspects to some degree. Together they contain examples of applications of most of the perspectives to conceptual modeling as discussed in Chapter 5 although none of them can be said to cover all perspectives in an satisfactory way, as will be described below.

7.1 OMT

Traditionally OMT (Object Modeling Technique) [271] is regarded as an object-oriented approach, but as many such approaches it combines features from several existing languages and uses them from an object-oriented perspective.

7.1.1 Languages for Conceptual Modeling

OMT have three modeling languages: the object modeling language, the dynamic modeling language, and the functional modeling language.

Object modeling language

This describes the static structure of the objects and their relationships. It is a semantic data modeling language. The vocabulary and grammar of the language are illustrated in Figure 7.1.

- a) Illustrates a class, including attributes and operations. For attributes, it is possible to specify both data type and an initial value. Derived attributes can be described, and also class attributes and operations. For operations it is possible to specify an argument list and the type of a return value. It is also possible to specify rules regarding objects of a class, for instance by limiting the values of an attribute.
- b) Illustrates generalization, being non-disjoint (shaded triangle) or disjoint. Multiple inheritance can be expressed. The dots beneath *superclass2* indicates that there exist more subclasses. It is also possible to indicate a discriminator (not shown). A discriminator is an attribute whose value differentiates between subclasses.
- c) Illustrates aggregation, i.e. part-of relationship on objects.
- d) Illustrates an instance of an object and indicates the class and the value of attributes for the object.
- e) Illustrates instantiation of a class.
- f) Illustrates relationships (associations in OMT-terms) between classes. In addition to the relationship name, it is possible to indicate a role-name on each side, which uniquely identifies one end of a relationship. The figure also illustrates propagation of operations. This is the automatic application of an operation to a network of objects when the operation is applied to some starting object.
- g) Illustrates a qualified relationship. The qualifier is a special attribute that reduces the effective cardinality of a relationship. One-to-many and many-to-many relationships may be qualified. The qualifier distinguish among the set of objects at the many end of an relationship.
- h) Illustrates that also relationships can have attributes and operations. This figure also shows an example of a derived relationship (through the use of the slanted line).
- i) Illustrates cardinality constraints on relationships. Not shown in any of the figures is the possibility to define constraints between relationships, e.g. that one relationship is a subset of another.
- j) Illustrates that the elements of the many-end of an relationship are ordered.
- k) Illustrates the possibility of specifying ternary relationships.

An example that illustrates the use of main parts of the languages is given in Figure 7.2 indicating parts of a structural model for a conference system. The model is made so that it is easier to compare it with the similar model in Section 7.4.

Dynamic modeling language

This describes the state transitions of the system being modeled. It consist of a set of concurrent state transition diagrams. The vocabulary and grammar of the language is illustrated in Figure 7.3. The standard state transition diagram functionality is illustrated in Figure 7.3a) and partly 7.3 b), but this figure also illustrates the possibility of capturing events that do not result in a state transition. This also includes entry and exit events for states. Figure 7.3c) illustrates an event on event situation, whereas Figure 7.3d) illustrates sending this event to objects of another class. Figure 7.3e), 7.3f), and 7.3g) shows constructs similar to those found in Statecharts [124] to address the combinatorial explosion in traditional state transition diagrams. Not shown in the figure are so called automatic transitions. Frequently, the only purpose of a state is in this language to perform a sequential activity. When the activity is completed a transition to another state fires. This procedural way of using a state transition diagram is somewhat different from the traditional use.

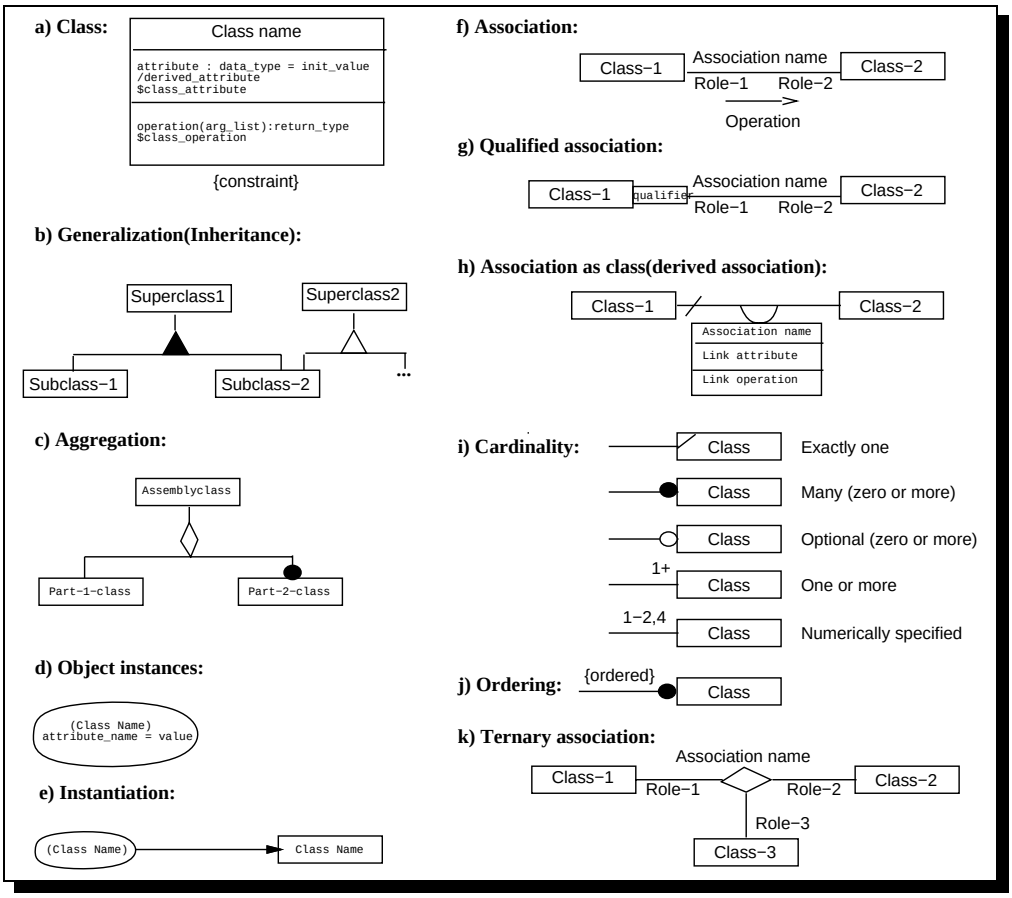

Figure 7.1: Symbols in the OMT object modeling language

Functional modeling language

This describes the transformations of data values within a system. It is described using data flow diagrams. The notation used is similar to traditional DFD as illustrated in Chapter 5.2.2, with the exception of the possibility of sending control flows between processes, which are signals only. Actor corresponds to objects as sources or sinks of data.

7.1.2 Methodology

The OMT methodology is divided into three phases; analysis, system design, and object design. The input to analysis is the problem statement and the output is a formal model that identifies the objects and their relationships, the dynamic flow of control and the transformation of data through the system.

The conceptual models are developed primarily in the analysis phase. It consists of the following steps:

- Write or obtain an initial description of the problem in natural language.
- Develop an object model:
  - Identify object classes.
  - Begin a data dictionary containing descriptions of classes, attributes, and relationships.

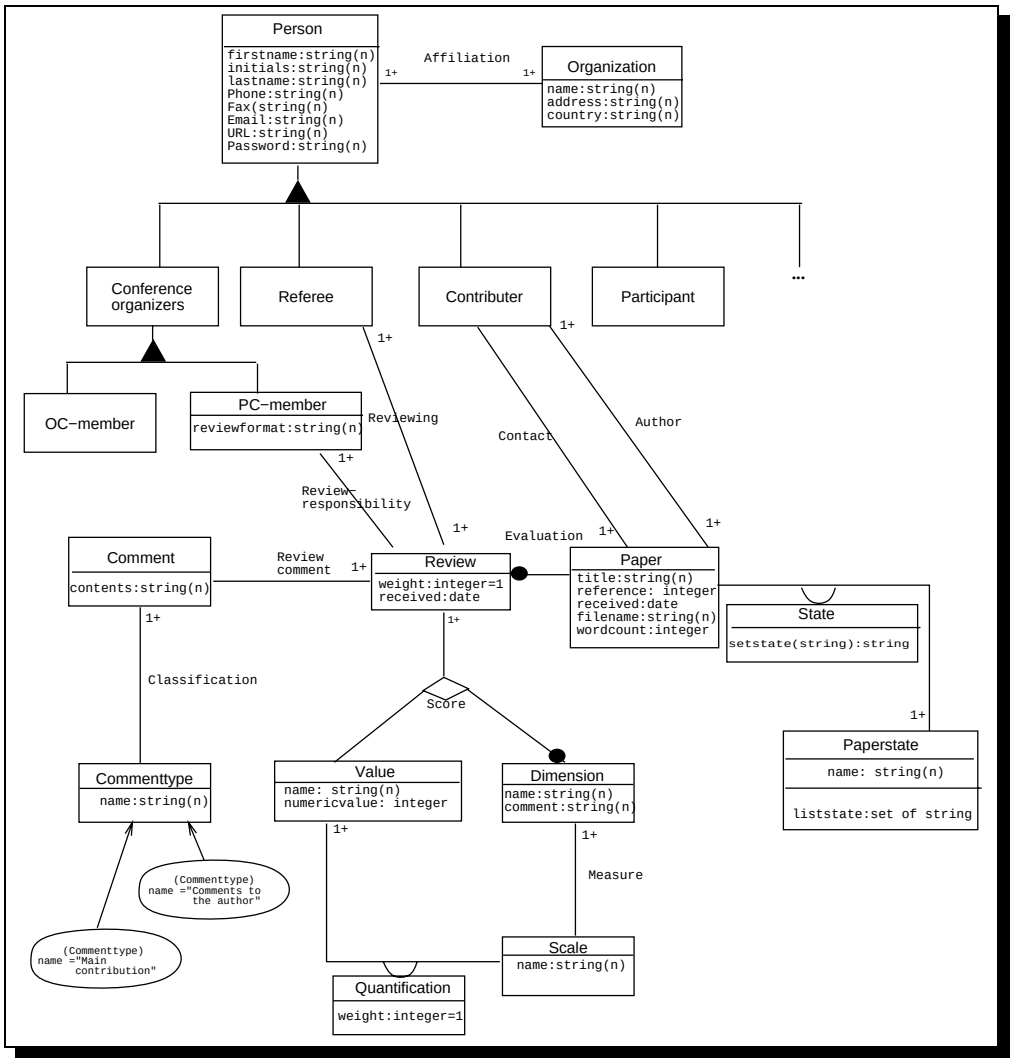

Figure 7.2: Example of an OMT object model

  - Add relationships between classes.
  - Add attributes for objects and links.
  - Organize and simplify object classes using inheritance.
  - Test access paths using scenarios and iterate the above as necessary.
  - Group classes into modules, based on close coupling and related functionality.
- Develop a dynamic model:
  - Prepare scenarios of typical interaction sequences.
  - Identify events between objects and prepare an event trace for each scenario.
  - Prepare an event flow diagram for the system.
  - Develop a state diagram for each class that has important dynamic behavior.
  - Check for consistency and completeness of events shared among the state diagrams.
- Develop a functional model:
  - Identify input and output values.
  - Use data flow diagrams as needed to show functional dependencies.
  - Describe what each function does.

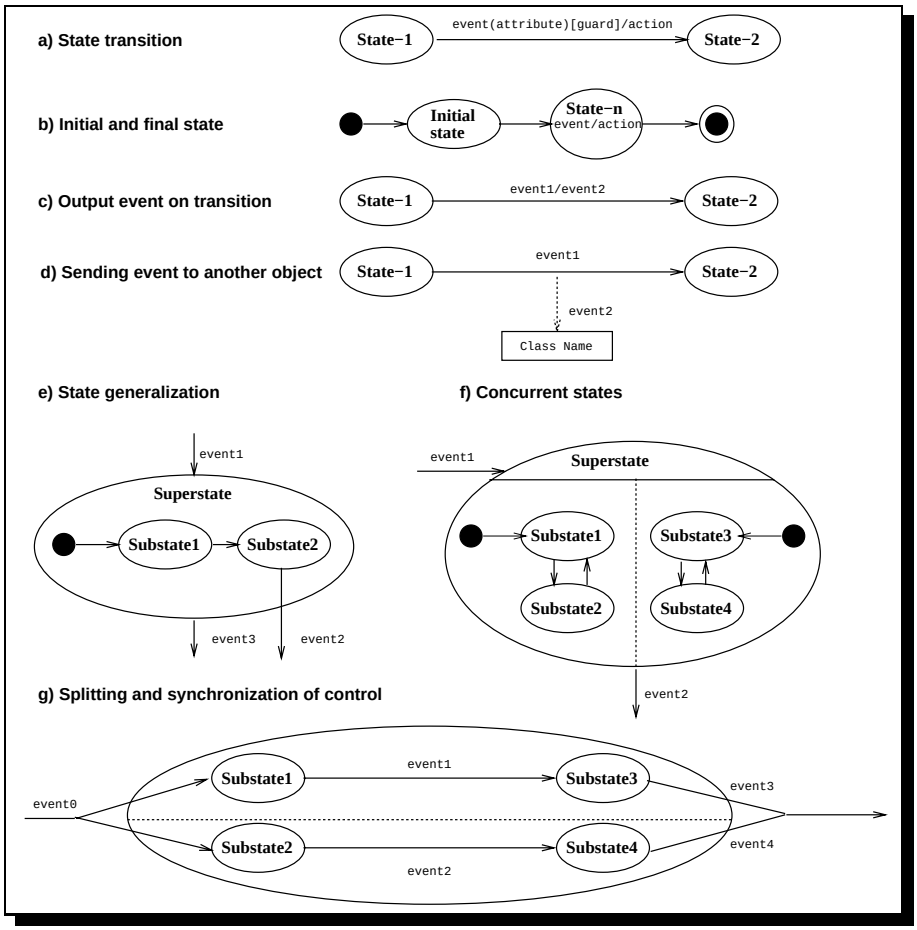

Figure 7.3: Symbols in the OMT dynamic modeling language

  - Identify constraints.
  - Specify optimization criteria.
- Verify, iterate, and refine the models:
  - Add key operations that were discovered during preparation of the functional model to the object model. Do not show all operations during analysis as this would clutter the object model; just show the most important operations.
  - Verify that the classes, relationships, attributes, and operations are consistent and complete at the chosen level of abstraction. Compare the three models with the problem statement and relevant domain knowledge, and test the models using scenarios.
  - Develop more detailed scenarios (including error conditions) as variations on the basic scenarios. Use these what-if scenarios to further validate the three models.
  - Iterate the above steps as needed to complete the analysis.

The next stage, system design, involves deciding on the organization of the system into subsystems and the allocation of subsystems to hardware and software components. Finally, object design involves further refinement of the initial models to address the requirements of an execution environment.

Versant Object Technology has released OMTool, which supports Rumbaugh's notation. OMTool is basically a graphical editor for constructing object models [332]. More recently, both IDE and Cadre Technologies have issued more advanced modeling tools to support OMT [249], and

it is these that we will refer to in the evaluation below.

7.1.3 Evaluation

The evaluation is primarily based on [141, 249, 332] in addition to [271].

Classification of OMT

- Weltanschauung: Objectivistic.
- Coverage in process: Primarily focused on (main parts of) development.
- Coverage in product: A single application system.
- Reuse: Partial support of development for reuse.
- Conceptual modeling: Methodology based in a large degree on active use of conceptual models.

Language quality

- Domain appropriateness:
  Compared to the object-model in Figure 5.14, OMT is one of the object-oriented modeling languages that has best coverage [332]. It supports the object and structural perspective well. It also supports the behavioral and functional perspective, but the perspectives are not clearly integrated. It can also be stated that both the support of the functional and behavioral perspective is taken better care of in other languages covering these perspectives(e.g. not all the aspects of Statecharts are included in the dynamic modeling language, and the functional modeling language is basically traditional DFD). Regarding rules, these are partly supported through constraints related to classes in the object-model, with the restrictions this have [137]. Rule-hierarchies are not supported, and neither are speech-act modeling. Actors and roles can be modeled using objects and classes, but there are no constructs for differentiate these from other classes, with the exception of the external actor in the functional model. Whereas static and dynamic aspects can be modeled, no explicit support of temporal aspects is given.
  The languages have both general and some very specific constructs, and are composable in a similar way as traditional data, process, and state transition modeling languages, which means that the composability is somewhat restricted. The languages can be flexible in precision, although the functional modeling language has the traditional lacks of DFD regarding formality.
- Participant knowledge appropriateness:
  The basis of OMT is a set of well known languages: DFD, ER, and state-transition diagrams. On the other hand the object-oriented emphasis of how these are applied might hinder the use at least initially, although less than with purer object-oriented approaches. It is not possible to model inconsistencies explicitly.
- Participant interpretation enhancement:
  The objects, processes, and states have very similar representation. The symbol used for aggregation is similar to the symbol use for ternary relationships. Actors are used both for actors and roles (as is usual in DFD). Cardinalities are shown in three different ways: As a set of numbers, as a filled or open dot, or as nothing (the last situation can give the impression that the diagram is not finished). Relationships are shown in four different ways,

according to if they are binary or n-ary (n¿2) and if they have attributes or not.
The way of indicating attributes and methods within the class can make symbols visually complex.
In the object model, emphasis is given to classes (and objects) through size which seems sensible. Emphasis is also given to a special form of cardinality and overlapping membership in specialization through blackness. Since one insert attributes and methods in classes and objects, these will differ in size. The positive aspect of this is that one have fewer symbols and types of symbols in the object model. Similarly can be said about the introduction of the union-notation for the state-diagrams, where the improved expressive economy is more important than the negative aspect of non-uniform size.

- Technical actor interpretation enhancement:
  All models have well-defined syntax. Both the object and dynamic modeling language have well-defined semantics.

Potential for creating models of high quality

When referring to tool-support below, we primarily refer to what can be found in the tool made by IDE (StP).

- Physical quality: Several tools exist for the creation and storage of models. Both the IDE and the Cadre-tool apply an adaptable meta-model so it is possible to add additional features. The repository also support standard although not very advanced database-functionality.
- Syntactic quality: Checking scripts can be invoked to report on violation of inheritance rules and cyclic inheritance. Incomplete and inaccurate models can be stored. Checks for consistency between diagrams are supported.
- Semantic quality: There is no explicit support of requirements capture and traceability. The tools support updating of models. The methodology suggest a set of guidelines for achieving complete and valid models.
- Pragmatic quality: It is possible to create code frames and database schemas from the models, a submodel, or a class, thus supporting the creation of prototypes. Detailed code must be added manually. The tool provides help facilities for explaining the purpose and correct usage of the modeling languages.
- Social quality: Not addressed specifically.
- Knowledge quality: Not specifically supported.

7.2 Tempora

Tempora [202] was an ESPRIT-3 project that finished in 1994. It aimed at creating an environment for the development of complex application systems. The underlying idea was that development of a CIS should be viewed as the task of developing the rule-base of an organization, which is used throughout development. Hence the rule perspective is dominant.

7.2.1 Languages for Conceptual Modeling

Tempora has three closely interrelated languages for conceptual modeling. ERT [216, 307], being an extension of the ER language, PID [116, 307], being an extension of the DFD in the SA/RT-tradition, and ERL [214, 307], a formal language for expressing the rules of an organization.

The models shown in this section are taken from the “Sweden Post Case Study” [306]. The case study is described further in Appendix A.

The ERT language

The basic modeling constructs of ERT are: Entity classes, relationship classes, and value classes. The language also contains the most usual constructs from semantic data modeling [253] such as generalization and aggregation, and derived entities and relationships, as well as some extensions for temporal aspects particular for ERT. It also has a grouping mechanism to enhance the visual abstraction possibilities of ERT models. The graphical symbols of ERT are illustrated in Figure 7.4. An example is shown in Figure 7.5.

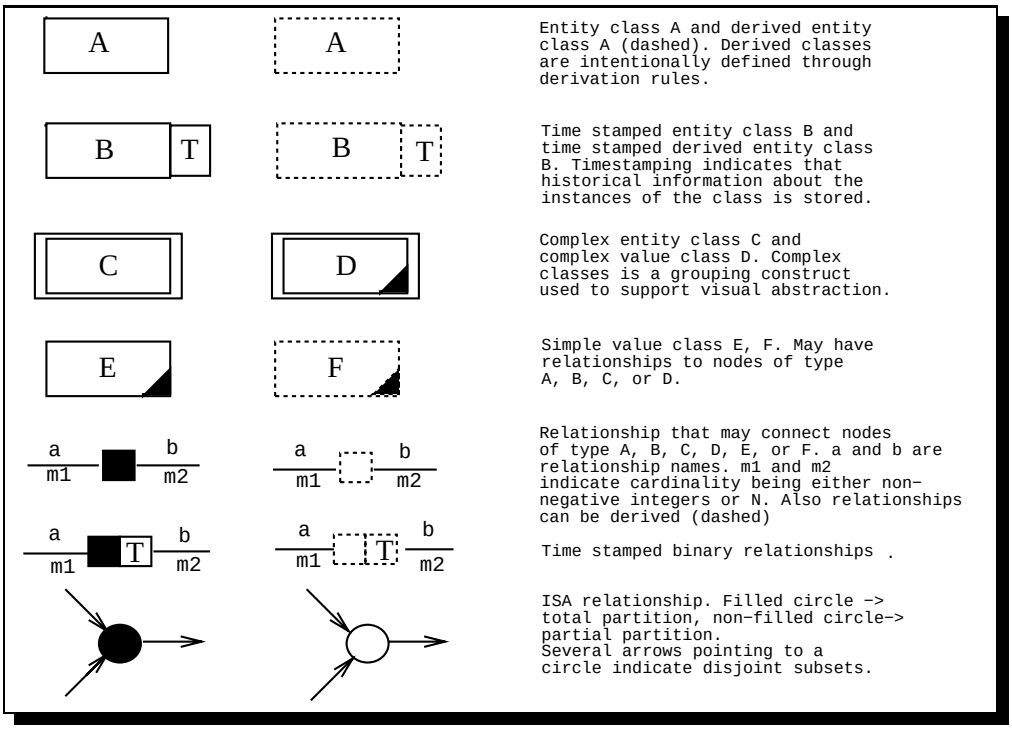

Figure 7.4: Symbols in the ERT languages

The PID language

This language is used to specify processes and their interaction in a formal way. The basic modeling constructs are: processes, ERT-views being links to an ERT-model, external agents, flows (both control and data flows), ports, and timers, acting as either clocks or delays. The language is similar to the one used in PPP [116] described in Chapter 7.4. The graphical symbols of PID’s are illustrated in Figure 7.6.

The top level PID of the Sweden Post case study is shown in Figure 7.7. The ports are omitted from the example. Decomposed processes are shown with a bold border.

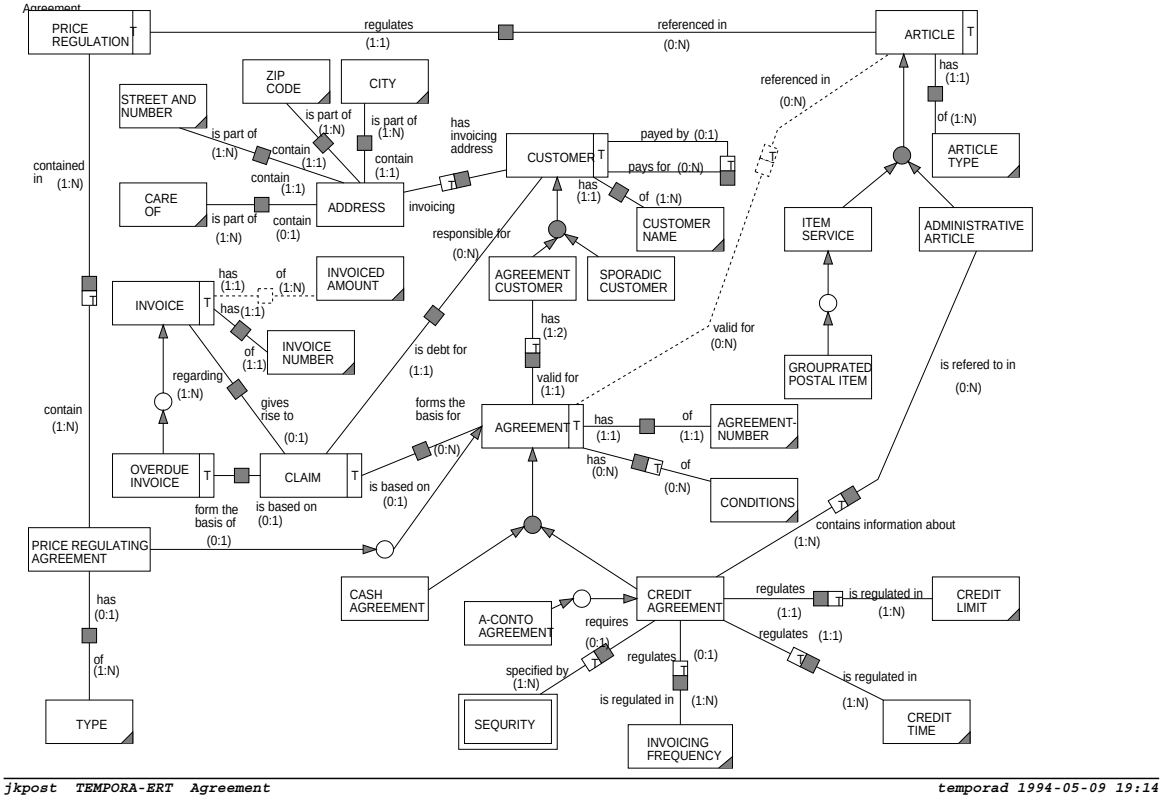

Figure 7.5: Example of an ERT model

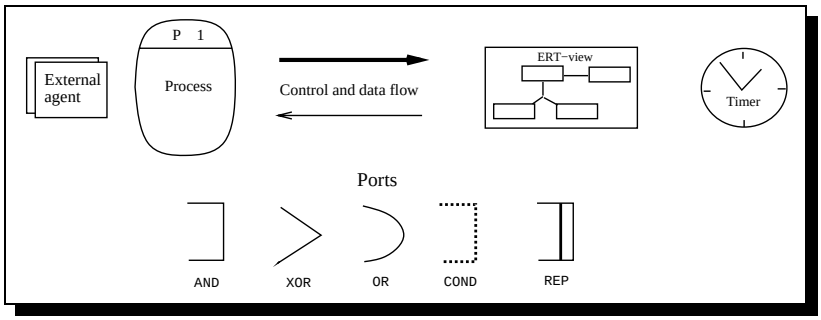

Figure 7.6: Symbols in the PID language

The external rule language (ERL)

The ERL is based on first-order temporal logic, with the addition of syntax for querying the ERT model. The general structure of an ERL rule is as follows:

> when *trigger* if *condition*, then *consequence* else *consequence*.

- *trigger* is optional. It refers to a state change, i.e. the rule will only be enabled in cases where the trigger part becomes true, after having been previously false. The trigger is expressed in a limited form of first order temporal logic.
- *condition* is an optional condition in first order temporal logic.
- *consequence* is an action or state which should hold given the trigger and condition. The consequence is expressed in a limited form of first order temporal logic. The 'else' clause indicates the *consequence* when the condition is not true, given the same trigger.

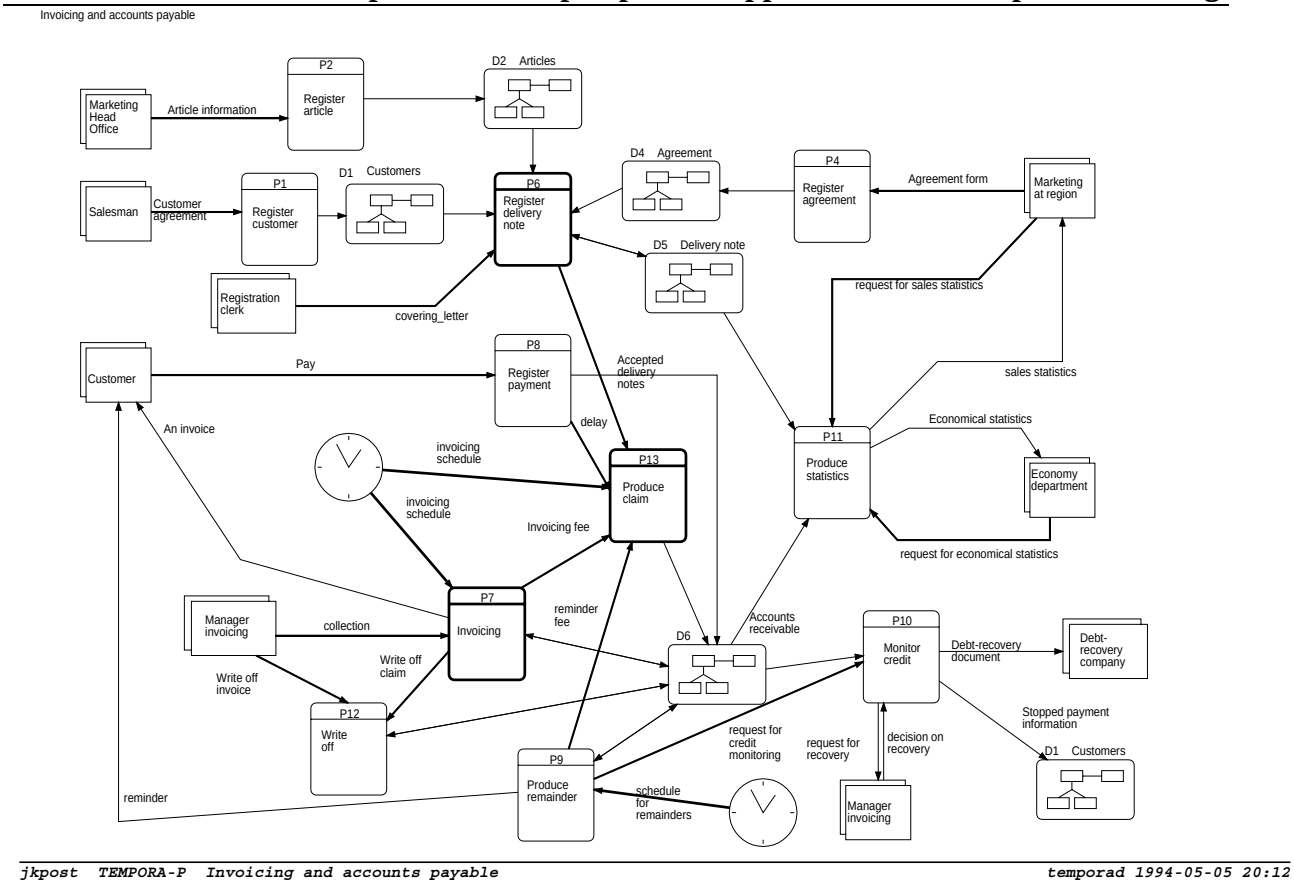

Figure 7.7: Example of a PID model

ERL-rules have both declarative and procedural semantics. To give procedural semantics to an ERL-rule, it must be categorized as being a *constraint*, a *derivation rule*, or an *action rule*. In addition it is possible to define *predicates* to be able to simplify complex rules by splitting them up into several rules.

The rule can be expressed on several levels of details from a natural language form to rules which can be executed. Some examples of the latter are given below.

- *Constraints* express conditions on the ERT database which must not be violated. The example reads that if a price regulation is valid for only one customer, it must not refer to an article of type group related postal item.

```
if  price_regulation.X
and number_of {C for_which price_regulation.X contained_in
               price_regulating_agreement valid_for
               agreement_customer.C} = 1
and price_regulation.X regulates article.A
then not grouprated_postal_item.A
and  article.A has article type <> ‘‘grouprated_postal_item’’
```

- *Derivation rules* express how data can be automatically derived from data that already exist. The example below shows how the derived (dashed) relationship in Figure 7.5 is derived.

```
if   credit_agreement.X contains_information_about
     administrative_article.Y
then agreement.X valid_for article.Y
```

- *Action rules* express which actions to perform under what conditions. Action rules are typically linked to atomic processes in the process model, giving the execution semantics for the processes as illustrated i Figure 7.8. A detailed treatment of the relationship between processed and rules is given in [215, 282].

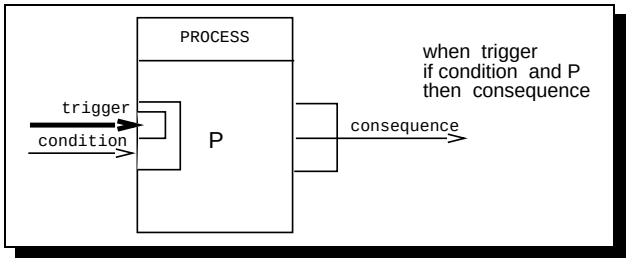

Figure 7.8: Relationship between the PID and ERL languages (from [179])

The rule below is one of the rules connected to process P9, *produce reminder* in Figure 7.7. The rule reads:
When the reminder schedule starts, check for overdue invoices and send a reminder to the customers responsible. The rule takes care of the case where no fee is issued. Another rule takes care of the opposite case.

```
when  schedule_for_reminder(_)
if    overdue_invoice
                [has invoiced_amount.A
                 has invoice_number.IN,
                 regarding claim is_debt_for customer.Cu
                        [has customer_name.CuN,
                         has_invoicing_address address.Addr
                                [contain street_and_number.SAN,
                                 contain zip_code.Z
                                 contain city.C
                                ]
                        ]
                ]
and   agreement_customer.Cu has agreement
      has conditions = ''no_reminder_fee''
and  only_one_of(address.Addr contain care_of.CO,CO =={})
then  reminder(IN,A,CuN,SAN,Z,C,CO)
```

Temporal aspects: The main extension in ERL compared to other rule-languages is the temporal expressiveness. At any time during execution, the temporal database will have stored facts not only about the present time, but also about the past and the future. This is viewed as a sequence of databases, each associated with some *tick*, and one may query any of these databases. ERL rules are always evaluated with respect to the database that corresponds to the real time the query is posed.

A time point identifies a certain tick during any day. A set of shorthand names are provided to avoid the need to calculate explicitly commonly used time point values. A time point is a short-

hand for a time interval of one tick duration. Time points can be compared and computed, and it is possible to specify that a certain expression should be valid at, after, or before a time point. In addition it is possible to specify longer time intervals. [t1,t2] indicates the continuous series of ticks starting at t1, and lasting up to and including t2. It is possible to specify that an expression is valid during an interval, and the language also supports operators to express the possible relationships between two intervals.

In addition to linking PID to ERT-models and ERL-rules to ERT-models and PIDs, one have the possibility of relating rules in rule hierarchies. The relationships available for this in Tempora are [281, 286]:

- *Refers-to:* Used to link rules where definitions or the introduction of a necessary situation can be found in another rule.
- *Necessitates* and *motivates*: Used to create goal-hierarchies.
- *Overrules* and *suspends*: These deal with exceptions. If an action is overruled by another rule, then it will not be performed at all, whereas an action which is suspended, can be performed when the condition of the suspending rule no longer holds. With these two relations, exceptions can be stated separately and then be connected to the rules they apply to. This provides a facility for hiding away detail, and at the same time obtaining the necessary exceptional behavior when it is needed.

7.2.2 Methodology

The defined phases in Tempora are planning, development, and operation and maintenance. Focus is on the development phase.

Development is divided into several phases:

- Development of conceptual model of the organization. The behavior and structure of the organization including both computerized and manual functions in the organization are described and analyzed and requirements to a new system are stated.
- Development of conceptual model of the CIS. The behavior and structure of the target system are described and analyzed. Everything needed for an executable specification should be provided.
- Development of design specification of the CIS. The specification is extended to include detailed design decisions, and computational semantics are specified.
- Run-time generation. A largely automated process based on the design.

Proposing a set of phases does not mean that an approach based on the waterfall methodology with disjoint phases performed in a strict sequence is promoted. Any project model can supposedly be overlaid the proposed approach in order to meet the management requirements of a development project.

Development of a conceptual model of the organization

Objectives in this phase are to get a better understanding of the area. Some of the documents one take into considerations are: business ideas, organizational charts, and organization-wide IS plans. If some of this is not available, it must be gathered from users or other stakeholders.

The modeling takes place according to a circular model of three steps:

- Initial analysis by interviewing stakeholders and reading documentation.
- Modeling seminars together with stakeholders.
- Analysis of result and follow-up discussions with stakeholders.

As illustrated in Figure 7.9, the modeling process is repeated until satisfactory results are gained.

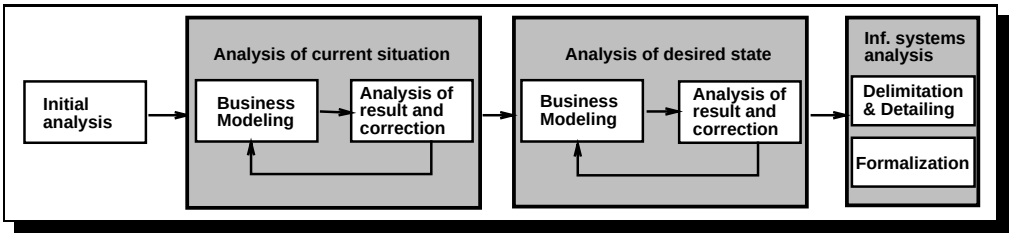

Figure 7.9: The main phases of requirements specification in Tempora

The modeling is divided into five not necessarily sequential tasks.

- Initial problem modeling include goal modeling and simple ERT, PID, and ERL modeling.
- Develop a business-wide ERT-model. Only the main phenomena of the business are modeled. The whole ERT language is used except from derived entity classes, derived value classes, derived relationships, and timestamping.
- Develop PID specification of the business. All the symbols in the PID-language are used. The modeling should include decomposition of processes until sufficient understanding of the organization is achieved. Details of the functional specification are not included. Thus process logic description may be left out or specified declaratively by using simplified ERL-rules.
- Develop ERL specification of the business. All the constructs of the ERL are used. The rules should be declarative in the sense of having logical semantics but not operational semantics.
- Review the specifications. The specification are reviewed to ensure that they can be clearly understood by the participants, and that the models are syntactically and semantically correct. Furthermore one should ensure that the specification reflect the users' requirements to the future working of the organization.

Not all the participants in the team will be familiar with the modeling languages. Education in the vocabulary, grammar, and use of the modeling languages is regarded as essential. Additional advice for how to use the conceptual modeling languages in general are developed including the use of driving questions between models created in different interrelated languages [307].

Development of conceptual model of the CIS

The main tasks of this phase are to determine the automation boundary and to model the target system. Only the parts which are relevant for the CIS are of interest in this phase. Implementation descriptions are avoided. The transformations of the IS-model into the CIS model is illustrated in Figure 7.10.

Except of goal modeling, the same techniques are used for analyzing the CIS as for the analysis of the organization. CIS modeling involves the following, not necessarily sequential activities: Determine boundary of future CIS using the PID, develop ERT of CIS, develop PID of CIS, develop ERL of CIS, and reviewing the specification.

Development the design specification of the CIS

In this phase, the CIS-specification is elaborated as follows:

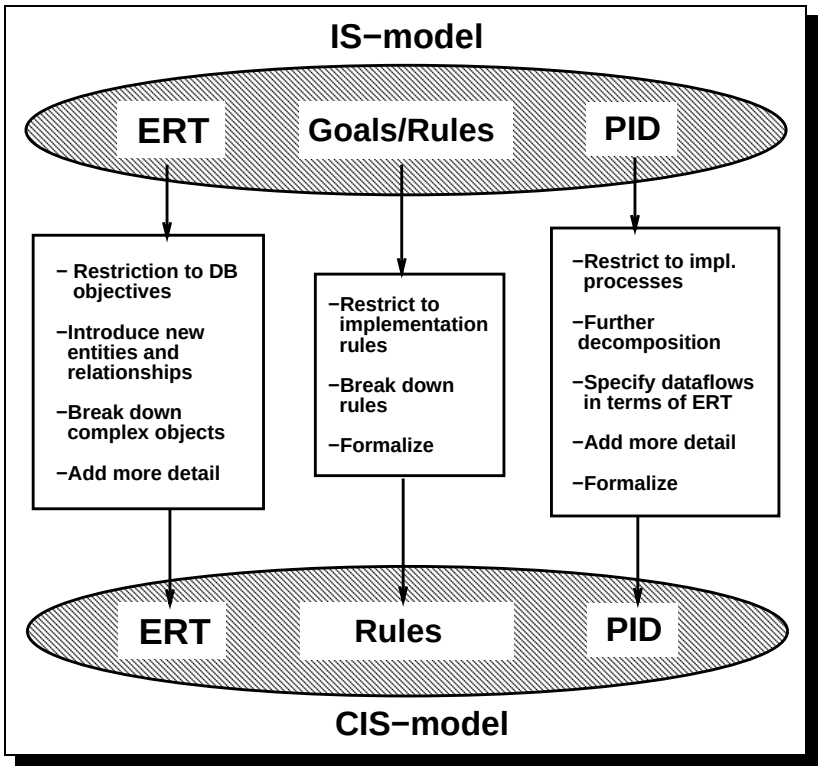

Figure 7.10: From IS-model to CIS-model in Tempora

- Derived entity classes, value classes, and relationships are added to the ERT-model.
- The logical ERL-model is transformed to a computational model. This is achieved by giving the rules operational semantics.
- The PID specifications are described at the lowest level of decomposition including implementation dependant details. In particular, details concerning external agents, flows, ERT views, and process logic are added.
- User interface design. Not specifically addressed in the methodology.
- Transaction boundaries are identified using PIDs describing the target system. In addition, the transactions are further detailed and optimization issues are addressed.

The specifications should be detailed enough so that the translation to a run-time system can be as automatic as possible.

Validation and verification of the system specification are done in all phases. Verification assures that the conceptual models are in accordance to the language model. The validation takes care of ensuring correspondence between the conceptual models and the participants view of the domain. This can be done with help of scenarios and animation. Tempora uses animation to show how the system react to various input to give an indication of the behavioral of the system. The result of the input is presented by graphical and textual components. The conceptual models do not need to be completed to visualize behavior of part of the system.

Run-time generation

The purpose of this phase is to generate an operational application system. The run time generator generate database schemas and executable code. TEQUEL rules are generated from the rule

model, and the database schema is generated from the ERT model and constraints expressed in ERL. The executable rules are fed into the rule manager which controls the actual execution of the rules.

7.2.3 Evaluation

Classification of Tempora methodology

- Weltanschauung: Objectivistic.
- Coverage in process: Primarily focused on development.
- Coverage in product: A single application system, but results from the business modeling parts of the methodology can also be useful on a portfolio level even if this is not stated specifically.
- Reuse: Apply generative reuse between phases, whereas component reuse is not specifically addressed.
- Conceptual modeling: Methodology based in a large degree on active use of formalizable conceptual models.

From the description, we see that Tempora is similar to the operational/transformational approach. What differentiate this from e.g. PAISLEY and GIST is that the conceptual modeling languages used are extensions of widely applied conceptual modeling languages such as ER and DFD.

Language quality

The below discussion is partly based on [282, 286].

- Domain appropriateness:
  Both the structural, functional, and behavioral perspectives are supported, whereas actors, organizational modeling, and speech act modeling are not well supported. The rule-perspective is especially well supported including the possibility of expressing temporal statements. The Tempora languages have been used in a number of case-studies, and these conclude that the languages have shown its feasibility with respect to expressiveness although certain perspectives are not covered as discussed above. Nor does Tempora differentiate between objects in the perceived world and data about these objects [282].
  The languages have both general and some very specific constructs, and are composable in a similar way as traditional data, and process modeling languages, which means that the composibility is somewhat restricted. The rule language has a very high composibility. The languages can be flexible in precision.
- Participant knowledge appropriateness:
  The basis of the Tempora languages is a set of well known languages: DFD, ER, and logic. For those familiar with this basis, the Tempora languages should be rather simple to learn and understand. However, the ERL suffers from many of the weaknesses of standard rule languages, being low level and computational. Whereas participants in the case-studies were reported to have small problems learning to use the process and data modeling languages, the detailed rule language was difficult to use and understand. In particular, rule-classification, constrains involving quantification, and temporal constructs appeared to cause confusion.
  Inconsistencies can partly be represented and dealt with using the overrules and suspends relationships between rules.

- Participant interpretation enhancement:
  There is no distinction between a relationship between two entity classes and a relationship between and entity class and a value class, thus symbol discrimination is not supported.
  In early process modeling, where the ERT-view is not necessarily linked to an ERT-model, this symbol is used inconsistently.
  The added expressiveness of the PID contributes to increase the complexity of large specifications. In particular, these problems are caused by complex port structure and a large number of flows. Another problem when it comes to symbolic simplicity is the ERT-view symbol.
  Large generalization hierarchies are difficult to comprehend. Factors such as inheritance rules, time stamping, and constraints expressed in ERL reduce the comprehensibility of the hierarchies.
  ERT has earlier been criticized for misguided emphasis [286]. Blackness is applied at the relationships as black squares in the middle of the link (or at the end for unary relationships) and at value classes as small black triangles. The increased degree of the value classes can also result in misguided emphasis.
  The use of ports often increase the problem of crossing lines in PIDs. The notation used for a similar construct in [183] can partly alleviate this problem.
  The external representation of the ERT-language is flat. Fragmentation of specification results because of space constraints. Although complex entities and value classes are included, the splitting up of large models is not supported and must be carried out in an ad hoc manner.
  ERL rules become very complex due to the high composibility. In spite of supporting user-oriented terms, the textual form has shown not to be effective for communication and understanding when the complexity grows. This is improved by linking rules to process and data models. In particular, the visualization of parts of rules e.g. explicit representation of control structures, contributes to increase understanding of complex rule structures and relationships between rules. This is an example where the use of graphical languages improve the perceptibility of a textual language. Another is the visual presentation of the goal-hierarchies.
  An aspect related to expressive economy is the black squares on the relationships. Undoubtedly, a relationship could also have been conveyed as a simple line between the two classes connected (as done, for instance, in OMT). Another aspect on expressive economy is the possibility to provide shared value classes, which positively can reduce the number of symbols in a simple diagram
  Aggregation using complex entity or value classes is conveyed in two ways:
  - The parts are put inside the aggregate.
  - There is a relationship drawn between each part and the aggregate.

  The second is obviously redundant, since a component is *necessarily* related to its aggregate. On the other hand it might be needed to represent cardinalities on the relationship.
- Technical actor interpretation enhancement: The modeling languages have a defined syntax and both an operational and a logical semantics. By applying the transaction specification construct also executability has shown to be good, taking into account that one have a rule-based run-time system working on top of a historical database.

Potential for creating models of high quality

- Physical quality: There exist a tool for model development based on a meta-case architecture. The tool have an underlying repository supporting basic database-functionality and distribution. It do not support more advanced physical quality support such as versioning, even if principles for this are discussed in accompanying literature. The tool is neither able to represent ports in PID-models and goal-hierarchies visually.
- Syntactic quality: The modeling languages have a formally defined syntax, enabling syntax checking. Syntax checking of ERL-rules is not implemented.
- Semantic quality. Tools for updating models exists, and it is felt that the proposed structuring mechanisms of the modeling languages help in the task of changing a large specification and keeping track of changes. This was felt to be a big problem when developing large specifications with early versions of the modeling languages. It exist possibilities for consistency-checking. The technique of driving questions based on the interconnection of the language is useful for improving completeness.
- Pragmatic quality: Supported through manual inspection, filtering, and simple animation. According to the methodology, this is not done until late in the development, i.e. there is little support for early prototyping. When models are sufficiently detailed also traditional execution of the model is possible by the creation of run-time systems.
- Social quality: Not specifically supported.
- Knowledge quality: The methodology do not discuss the aspects of knowledge quality or participant education in any depth, although the need for participant training both regarding the modeling languages and the modeling area is emphasized.

7.3 SAMPO

'SAMPO' is claimed to be an acronym for Speech-Act-based office Modeling aPprOach [13, 14, 15] [1]. In SAMPO, office activities are studied as a series of speech-acts. Two graphical languages are proposed for discourse analysis. They are supplemented with several tables that describe the elements of discourse in more detail.

7.3.1 Languages for Conceptual Modeling

The graphical languages for describing discourses are the discourse modeling language and conversation modeling language.

Discourse Modeling Language

A discourse model delineates discourse objects and their properties and relationships. It is a model of an institutionalized discourse and its comprehensive structure that is used to identify a discourse and its topic. The modeling language for discourses are represented in Figure 7.11.

Discourse models can distinguish the following:

- Entities, speech acts, and instrumental acts.

[1] In [14] the name has changed to "Speech Act-based information analysis Methodology with comPuter-aided tOols". This seems to be a good example of a "backronym" [98] where the acronym was created first, and the name second. 'Sampo' is most probably taken from the Finnish national poems of Kalevala, where Sampo is a machine that could produce anything you desired.

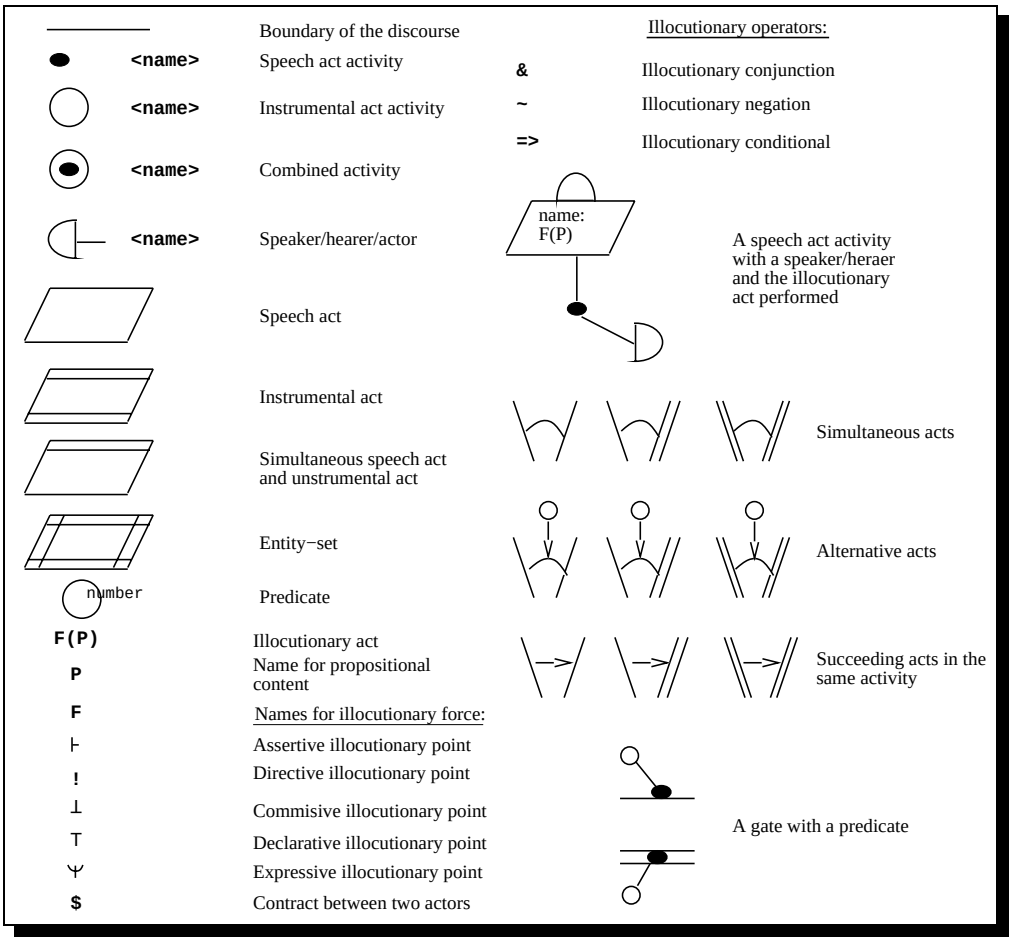

Figure 7.11: Symbols in the discourse modeling language (From [15])

- Activities.
- Components of speech acts and instrumental acts.
- Property predicates that describe context.
- Some components of the illocutionary force.
- Simple and complex illocutionary acts.
- Time schedules for commitments.
- Relationships among objects.
- Control predicates for the initiation and evolution of the discourse.
- The naming function that gives the discourse name.

Topicalization, i.e. those aspects of an utterance that are put into focus in a stage of a discourse, can be represented in a discourse model on two levels, both as entities and instrumental acts mentioned in speech acts, and as speech acts themselves.

To illustrate the modeling language, they apply parts of a conference example on the aspect of inviting and catering for speakers.

SAMPO provides several tables to describe the discourse informally. In the discourse type table, name, purpose, wanted effect, aim and ambiguities of the discourse is represented. In the next step one identify the main speech acts forming the discourse. The initial list of speech acts in this discourse is as follows (the illocutions are written in italics).

- 1. The organizing committee *commits* the organizers to inviting speakers on agreed subjects.
- 2. The organizing committee *commits* the organizers to keeping speaker costs within the boundary of the budget.
- 3. The organizer *invites* a speaker.
- 4. The speaker *accepts* the invitation.
- 5. The speaker *rejects* the invitation.

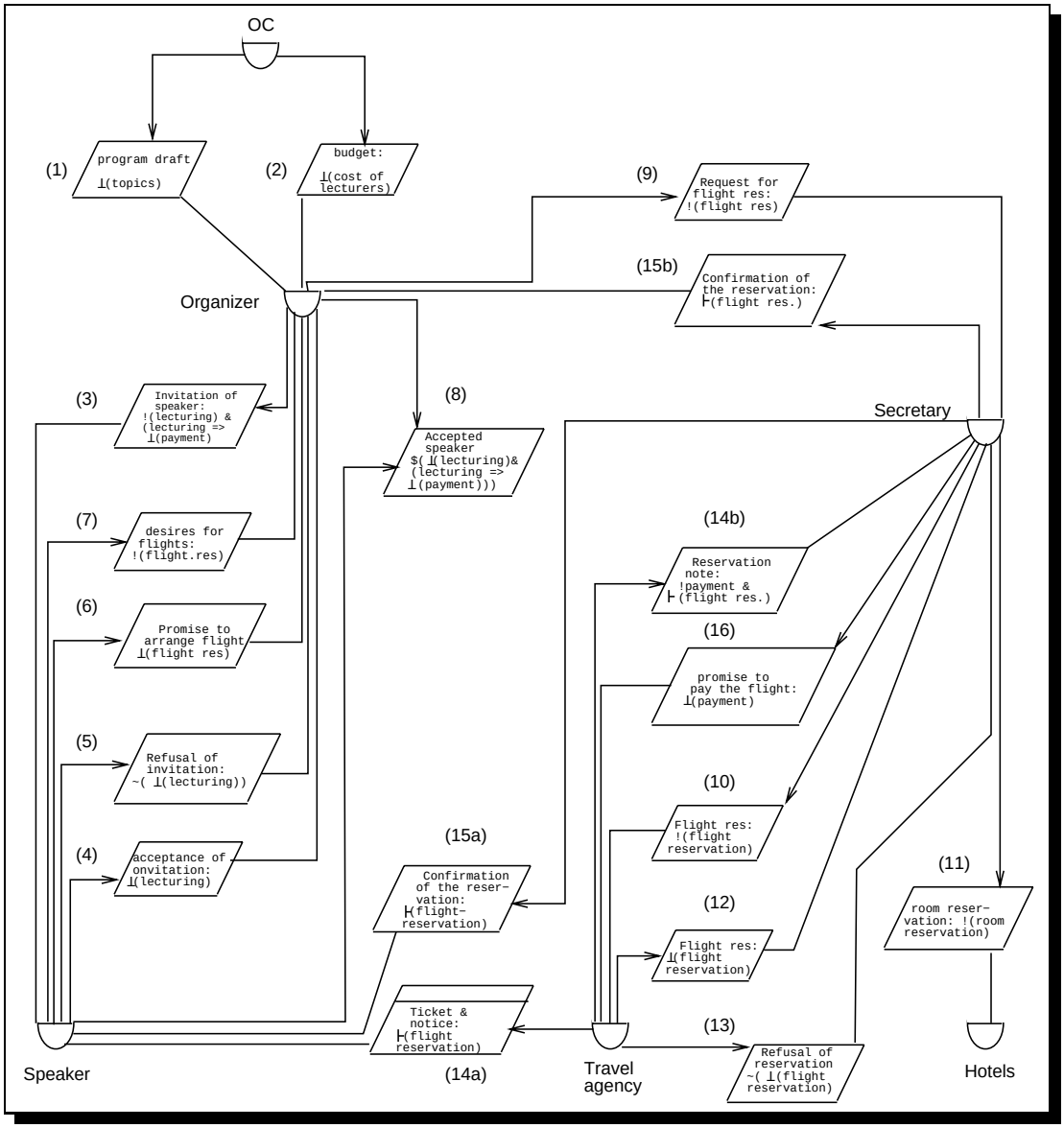

Figure 7.12: Example of a discourse model (From [15])

- 6. The speaker *promises* the organizer to make flight reservation himself.
- 7. The speaker *tells* the organizer that he wants travel arrangements to be made.
- 8. The speaker *prepares the lecturing contract* with the organizer.
- 9. The organizer *requests* the secretary to arrange the flights, and *informs* the secretary of the desires of the speaker.
- 10. The secretary *request* the travel agency to make the flight reservations.

- 11. The secretary *requests* the hotels to book rooms for the speakers.
- 12. The travel agency *promises* to reserve the flight.
- 13. The travel agency *refuses* to reserve the flight.
- 14. The travel agency *confirms* the reservation by sending the ticket (a) and invoice (b) to the secretary.
- 15. The secretary *confirms* the reservation with speakers and organizers.
- 16. The secretary *makes the commitment* to pay the invoice by the time it is due.

These speech acts are presented graphically in Figure 7.12. The success of the speech acts in this situation presupposes knowledge of the roles of the different actors. Their preparatory conditions may require, for example, that the secretary is in a position to make requests for flights.

Ultimately, the speech acts might ”produce” an instrumental act - the invited speaker gives a lecture at the conference. This is represented by the propositional content ”lecturing”.

From this overall model, one can select commitments concerning various discourse participants in different discourse segments, as, for example, those concerning the conference organizer in the invitation process. In Figure 7.13 this is illustrated by a discourse model that depicts the commitments made in this situation.

The actors can be further specified in informal tables indicating tasks and speech acts that the actor are involved in. Speech acts and activities are also further specified in informal tables specifying for speech acts volume, time and frequency, channel, and message form. For activities, purpose, positions, and goals of the activity are represented. Finally, predicate tables that describe mechanisms used to control the flow of discourse can be specified.

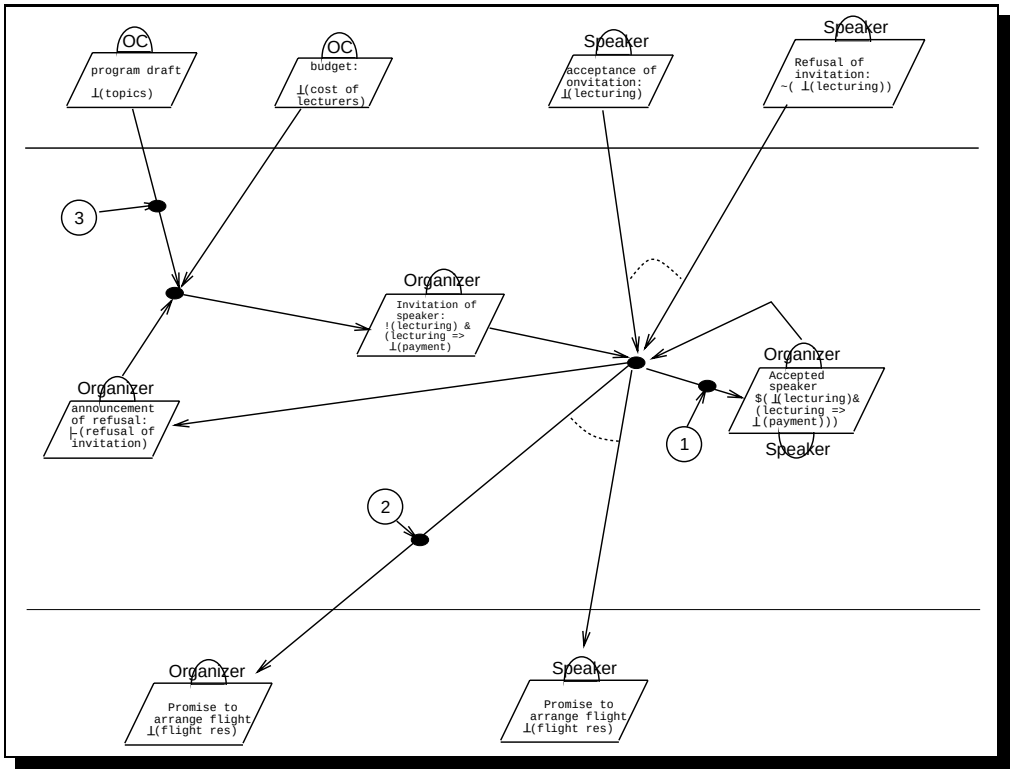

Figure 7.13: Detailed model of invitation of speaker(From [15])

Conversation modeling language

In clarifying the coherency of the discourse, one identify the success conditions for each discourse stage in a conversation model. The conversation model shows patterns of speech acts and instrumental acts and their dynamic interdependencies. The number of communicators are not limited to two. Conversation models can be used to establish that a discourse is well-formed in the following ways:

- For each speech act in the model, one show its conditions of success at each stage.
- For each stage in the conversation graph, one enumerate possible alternatives.
- For each stage one show how commitments are coordinated.

Conversation models describe:

- Stages in the conversation.
- Moves in the conversation.
- Implicitly mentioned speech acts that are needed to perform a speech act in the next stage.
- Property predicates describing the turn-taking sequence and topicalization.
- Predicates that restrict the sequential, parallel, and selective performance of acts.
- Control predicates.
- Gate predicates that enable the succeeding moves.
- The naming function that gives the name of the discourse process being analyzed.

The symbols of the conversation modeling language is given in Figure 7.14.

Figure 7.15 presents a conversation graph that describes the dynamics of commitments presented in Figure 7.13.

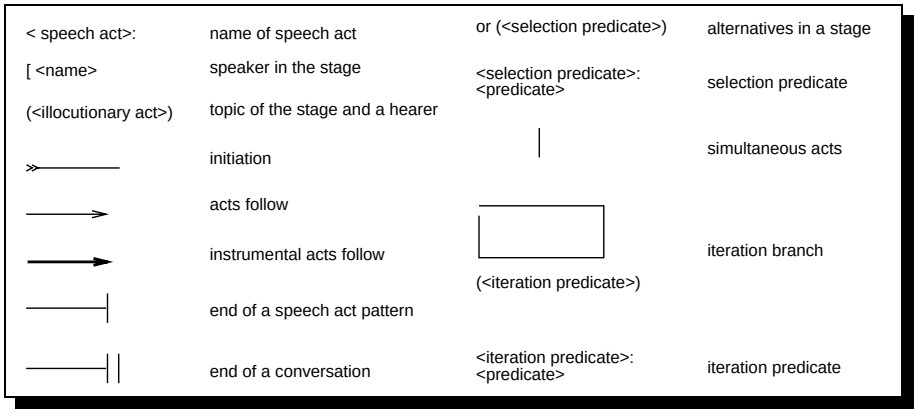

Figure 7.14: Symbols in the conversation modeling language(From [15])

Many aspects of coherency relate to specific features of transacting. For example, the request for flights cannot be made before the speakers accept the invitations and before they express their desire for flights (illocutionary conditional).

The study of discourse completeness analyzes conditions under which all discourse processes terminates. To test for completeness, one can transform a conversation model into a reachability tree.

By probing the coordination of commitments created, one can check what sort of commitments bind different actors during each stage of the discourse, how these evolves and by whom, and when the commitments are fulfilled. In SAMPO the method of studying coordination of commitments is to build a network in which one shows relationships between acts and evolution of

commitments. This is illustrated by attaching three operation to relationships between speech acts: Create (+), modify (#), or nullify (-).

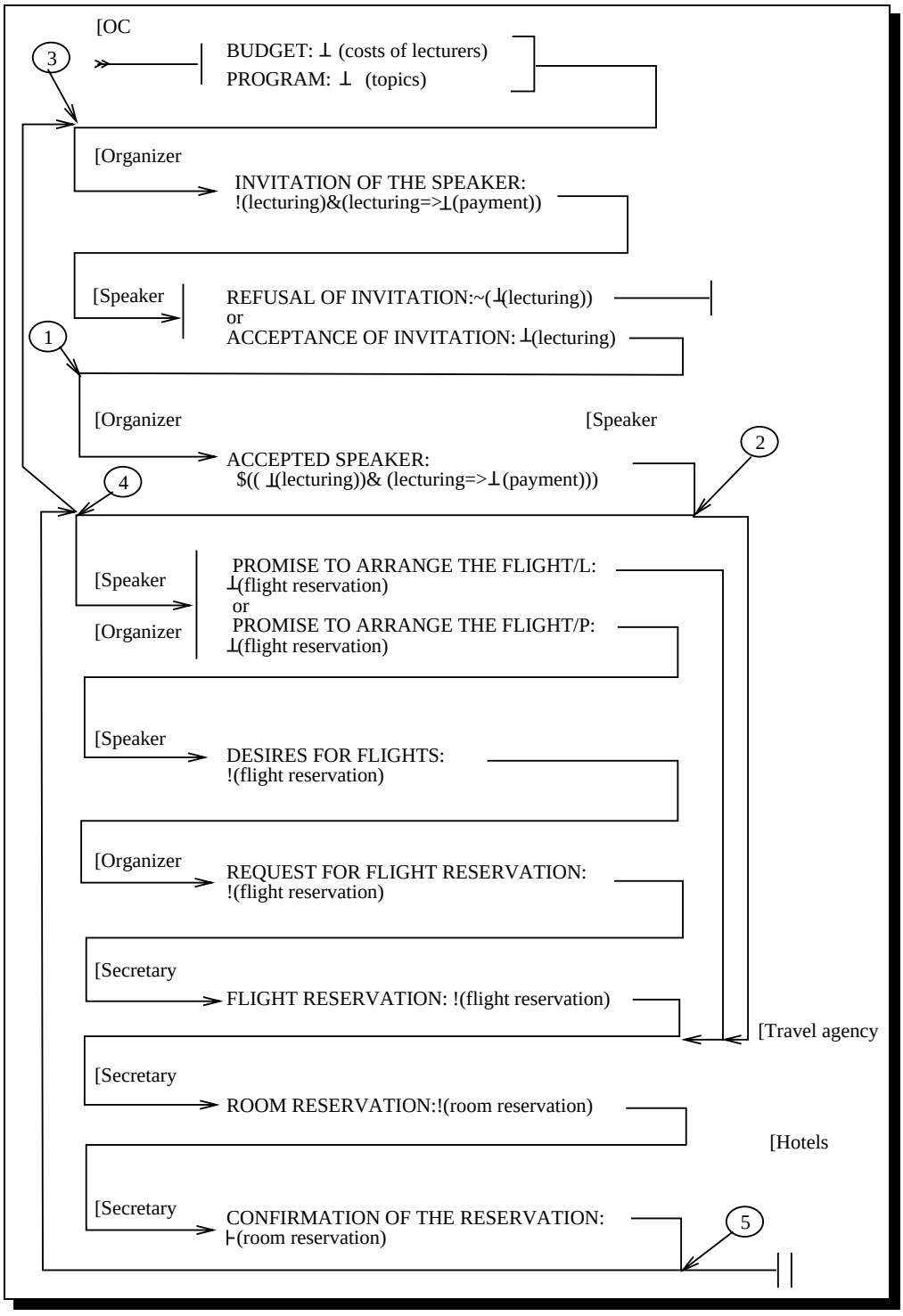

Figure 7.15: Example of a conversation model(From [15])

7.3.2 Methodology

SAMPO lacks the expression of technical aspects in its graphical presentation, but these are partly included in the tables. SAMPO covers only early phases of information systems development. Although some analysis support was illustrated above, the analysis methods in the current state of the model are mainly based on intuition and informal reasoning.

The overall development methodology has three phases:

- Change analysis, that tries to identify the problems and potential development areas.
- Information systems specification, where the selected (and alternatives for) information systems solutions are described.
- Information systems design, where the question dealing with cost effective design and implementation solutions are discussed.

According to the authors, SAMPO's most significant advantage compared to other office-modeling approaches is its communication-oriented conceptual structure. A disadvantage is that the structure is extensive. In the publications from 1992 the modeling approach was still at a preliminary stage, and where to undergo further development in areas such as: The elaboration of exceptions in conversations, a stricter definition of the modeling constructs, development of tools to support the methodology, and its application in field tests. We have found little work by the original authors on the approach from the last years.

7.3.3 Evaluation

Classification of SAMPO

- Weltanschauung: Constructivistic in its original motivation [107].
- Coverage in process: Development, primarily early phases.
- Coverage in product: Single application area or system.
- Reuse: Not addressed.
- Conceptual modeling: Methodology based in a large degree on active use of conceptual models in parts of the development.

Language quality

- Domain appropriateness:
  As described in [13] SAMPO provide high-level constructs to describe data/messages, processes, actors, and technology, which are the fundamental elements of an office according to [13]. We note that the description of detailed data-structures and rules are poorly supported, the same can be said about organizational structure. A pro compared to most other approaches is the explicit representation of speech acts. Goals can also be represented, although no goal-hierarchy is explicitly supported.
  The languages are fairly composable, and flexible in precision.
- Participant knowledge appropriateness:
  We have not seen any reports from larger case-studies using the approach to assess the problems of applying it and understanding it. We expect that similar problems with understanding the linguistically motivated parts of the language as reported in [41] might appear.
  When it comes to intuitivity, most of the symbols for different illocutionary points will be intuitive for persons familiar with formal logic. Some of the symbols used are similar to those applied originally by Searle [279]. It is not possible to model inconsistencies directly.

- Participant interpretation enhancement:
  We will here concentrate on the language for discourse modeling. The symbols used in this language are partly similar to those used for making A-graphs in ISAC [207] although they are used somewhat differently, i.e. the symbol indicating an instrumental act in SAMPO is used to indicate a real set in ISAC.
  Symbol discrimination is partly difficult, because of the different use of almost equal symbols (e.g. the use of the parallelograms)
  The symbols for instrumental act activity and predicate is equal, although the labeling will differentiate them. The symbol for speaker is different in the two languages, thus violating uniformness. The same applies to acts. Another breach of uniformity is how symbols are combined. Whereas a combined activity includes both the symbol for speech act activity and instrumental act activity, the symbol for a combined act is not in the same way a combination of the symbols for speech acts and instrumental acts.
  The use of emphasize seems well motivated according to the main perspective of the language, where acts and activities are emphasized with size and blackness respectively.
- Technical actor interpretation enhancement:
  The syntax for the modeling languages are formally defined, but not the semantics at least not in the newer descriptions of the languages. Thus executability is also limited. Earlier descriptions of the languages contain a more formal specification [210].

Potential for creating models of high quality

- Physical quality: See discussion on language quality above. Internalizability will be better supported when having proper tool-support, although it is difficult to know to what extent different database-functionality will be supported.
- Syntactic quality: Checking of completeness and coherency is put in this category, although this might also be used to improve the semantic quality.
- Semantic quality: Model updates similarly need support by a tool.
- Pragmatic quality: Not specifically supported.
- Social quality: Not specifically supported.
- Knowledge quality: Not supported.

7.4 PPP

PPP is an experimental CASE environment that has been developed at NTH, Norway. We will in this section give an overview of the PPP conceptual modeling languages based on [116, 265, 340].

7.4.1 Languages for Conceptual Modeling

PPP has four closely interrelated languages for conceptual modeling. ONER, being an extension of the ER language, PPM, being an extension of DFD in the SA/RT-tradition, PLD, a language for the specification of process logic, and the actor modeling language. In addition UID, a language for the modeling of graphical user-interfaces, is closely related to the other languages.

ONER

The symbols and vocabulary and some of the grammar rule of the ONER-language are illustrated in Figure 7.16. ONER is used to specify entities and relationships among entities. The constructs entity class and relationship class are used to express this. To express properties of the phenomena and possible operations on them, the constructs of data type and method are provided.

Relationships that can be defined between entity classes are illustrated in Figure 7.16II:

- *in_a_relationship* exists between an entity class and a relationship class. There are four possible types: *full_1, full_n, partial_1* and *partial_n*. *full* means that any entity in the class must be involved in at least one occurrence of the relationship whereas *partial* means that the involvement is not mandatory. *1* and *n* have the same meaning as in the ER language. The use of N-ary relationships is also similar to ER-modeling.
- Generalization. A generalization hierarchy can be formed. If an entity class E has subclasses $S_1, \cdots S_n$, these subclasses may *make up* E ($S_1 \cup S_2 \cdots \cup S_n = E$), or be *distinct* such that $\forall S_i, S_j \; S_i \cap S_j = \emptyset$, or both (a *partition* of E). Multiple inheritance is possible, although there are no mechanisms to support conflict resolution.

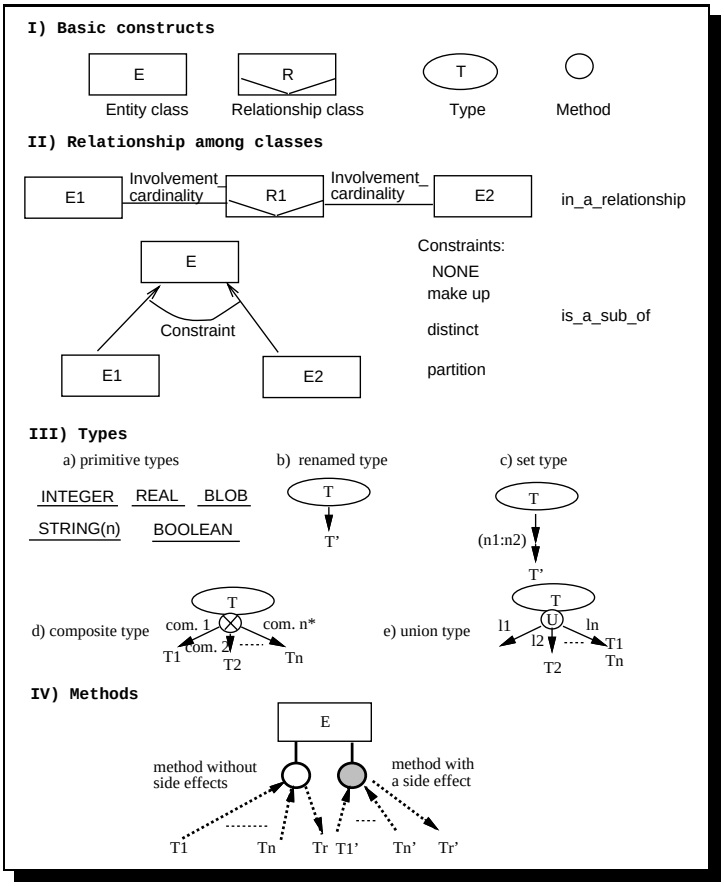

Figure 7.16: Symbols in the ONER language

Properties of an entity or a relationship can be expressed. The value of a property is an element of a *domain*. The domain is represented in ONER by *data types*. Several primitive data-types– *integer*, *real*, *string[n]*, *boolean*, and *blob* (Binary Large Object) are pre-defined(Figure 7.16IIIa). More complex data types can then be defined by the following constructors:

- *renaming* (Figure 7.16IIIb) : A type T is defined by renaming an existing type T'.
- *set* (Figure 7.16IIIc) : A type T is defined on the power set of the value set of type T'. Two numbers may be added to the definition for the minimum and maximum numbers of elements of a value of type T. The default is $(0, n)$.
- *compositing* (Figure 7.16IIIc) : A type T is a composite type with n components of types $T1, T2, \cdots Tn$. *com.1, ..., com.n* are the names of the components. "*" indicates an optional component.
- *union* (Figure 7.16IIIe) : A type T is the union of type $T_1, \cdots T_n$ with $label_1, \cdots label_n$, i.e,. any value of type T is either of type T_1, $\cdots$ or type T_n.

For an entity class or a relationship class, one can also define *methods* that define possible operations on the properties of the members of the class. A method is a function which has one or several input values as its parameters and one output value as its result. Two kinds of methods are distinguished according to whether a method updates the value of the properties. In Figure 7.16IV we give the graphical notations of the two kinds of methods. The implementation of methods may be given in a programming languages or be described in a PLD (see below).

In addition one can express other specific constraints, e.g., the limitations on the values of the attributes of some entities in a first order and many sorted language. Class attributes and derived entity classes can be supported by using SDL, a semantic data language for ONER. SDL is an SQL-style language which support data definition, manipulation and querying in terms of the ONER model.

In Figure 7.17 part of the data model of the IFIP conference is shown. This model represent the same situation as the OMT-model in Figure 7.2.

It is possible to construct scenarios of a developed model, being defined as a logically connected subset of the model. A scenario of a model can be made by performing a set of syntactically complete deletions of a model. It is possible to translate ONER models into a logical theory to enhance consistency checking.

PPM

The basic modeling components of PPM are illustrated in Figure 7.18.

- **Process**: A specified activity which can transform some inputs to outputs when it is invoked.
- **Store**: A place where a collection of data or material can be kept.
- **Timer**: Being either clocks or delays.
- **External agent**: A person, an organization, or another application system outside the scope of the IS, interacting with the it.
- **Flow**: A movement of *items* between external agents, processes, stores and timers. Items are the objects that are stored in stores, sent or received by external agents, or carried by flows. An item has a material aspect and a data aspect; one or both of these may be missing, in case the item is a signal. When a flow appears at the input or output of a process the following properties can be specified:
  - *Triggering:* This means that an input can only arrive when the process is not executing (idle), and the arrival of the triggering inputs in a legal combination will start the process. At the start of execution, the process will first receive all the triggering inputs.
  - *Terminating:* This means that an output can only be sent when the process terminates its execution. All the termination output flows in a legal combination will be sent out

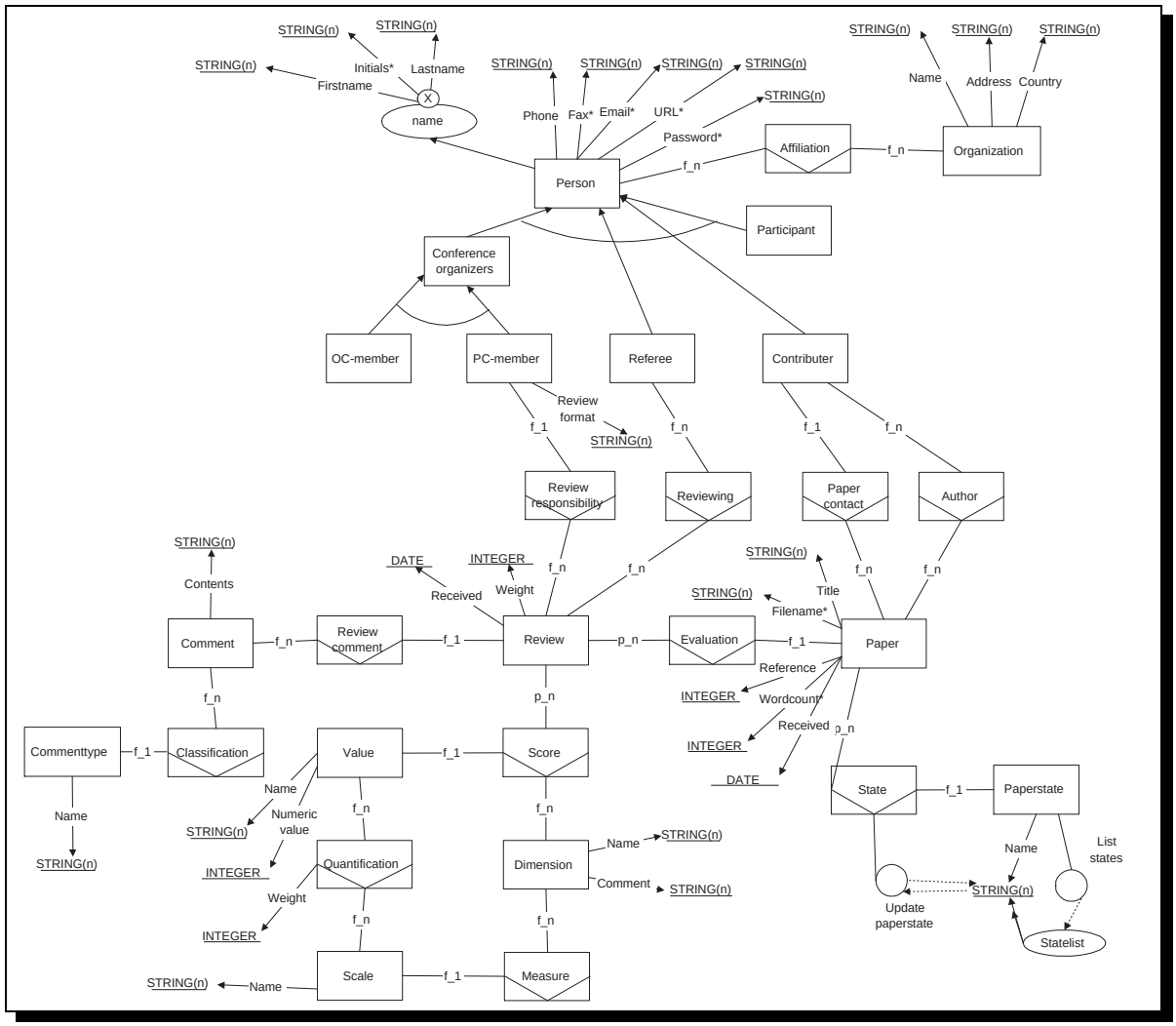

Figure 7.17: Example of an ONER-model

before the process changes its state to idle.

- *Singular flow*: The process will receive or send only one item on the flow during execution.
- *Repeating flow*: The process can receive or send more than one item on the flow during execution.
- *Conditional flow*: The process might, but do not need to receive or send an item on the flow during execution.

In addition to this *ports* which groups flows showing the logical combinations of items consumed or produced by a process when executing can be specified.

- AND port: All the members of the port are going to be received or sent during an execution of the process.
- XOR port: One and only one member of the port is going to be received or sent during the execution of the process.
- OR port: At least one member of the port is going to be received or sent during the execution of the process.

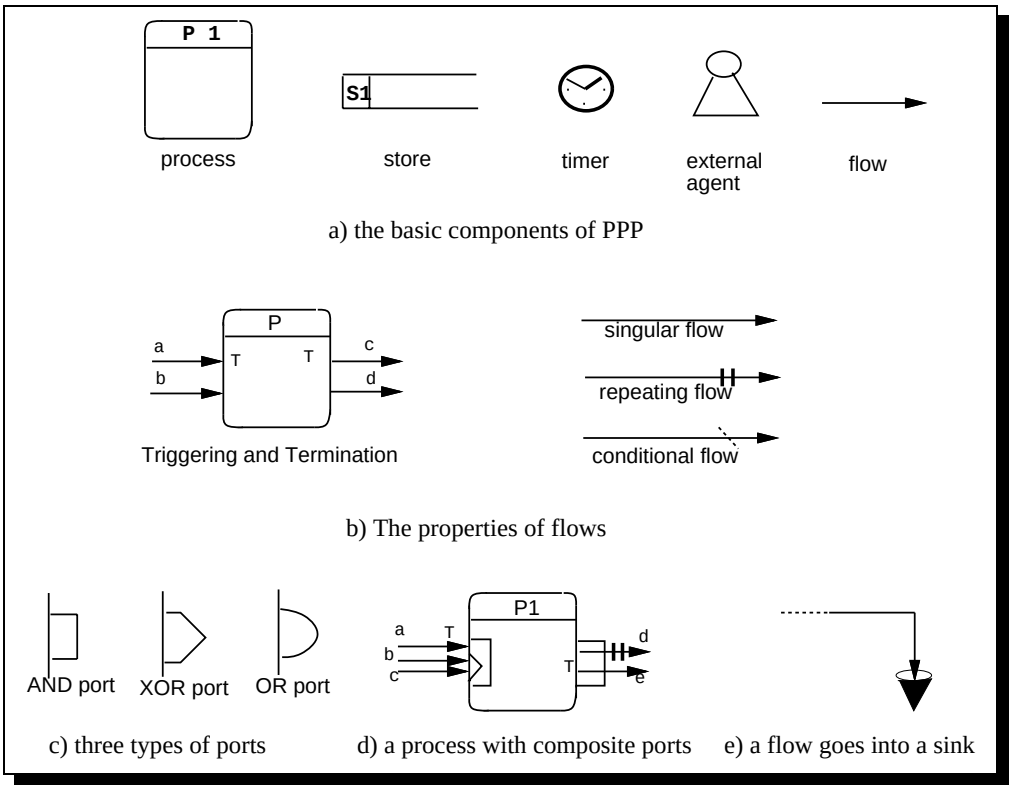

Figure 7.18: Symbols in the PPM language

This is a recursive definition, so on a process one can define *composite* port structures (Figure 7.18d). One is able to transform any port structure to a unique structure which is equivalent to the original port. The unique structure is of the form **xor(and(**$P_{11}, \cdots, P_{1m_1}$**),** $\cdots$**,and(**$P_{n1}, \cdots, P_{nm_n}$**))**, where every P_{ij} is a flow. This is called a *canonical port*, which is used in connection with consistency checking.

Normally a flow goes between components. A possible exception is that the item in a flow is lost. For specifying this case one has defined sinks (Figure 7.18*e*).

A PPM is a network of processes, stores, timers, sinks, and external agents connected by flows. Processes might be decomposed in additional PPM's.

A flow consists of output, channel, and input. An output delivers one of the resulting items of a process to its outside. A channel lets items pass through it. A channel might have a limited capacity. An input receives one of the items on a flow into a process. An input can receive items from more than one channel and an output can put items into several channels simultaneously.

Timers are of two kinds, clock and delays:

- **Clocks** are used to model events that are to occur at a specific moment in time. The flows connected to a clock are of three kinds:
  - On: Starts the clock. A clock may have zero, one or many on-flows. When switched on, the clock waits a specified clock interval, issues a clock signal, waits another clock interval, and so on until it is eventually turned off. Several signals might enter through the same on-flow starting new sequences, but will not remove an old one. When no on-flows are specified, the clock is turned on when the systems is started.

  - Off: Turns the clock off. The clock may have zero, one or many off-flows. It takes a new signal on an on-flow to restart the clock again.
  - Out: The clock may have one or more outputs. Items leaving a clock will enter a process, usually it is a signal triggering a new process execution.
- **Delays** are used to model events that are delayed with respect to time. A delay has at least two flows connected to it, one on-flow and one out-flow. These flows can be pure signal-flows or might contain data or material. In addition, the delay may have an off-flow. When an item is received on the on-flow, it will wait the specified delay, before forwarding the item on the out-flow, if the delay is not turned of in the meantime by the reception of a similar item on the off-flow. The delay might have several on, out and off-flows.

In the case of a simple delay or clock-interval such as "2 days"(relative time) or "Friday" (explicit time), this is indicated in the graphical symbol. For most timers, this will be sufficient, but if one wants to be able to specify more complex delays and clock intervals rules in a when-if-then form are suggested. Also external agents can be described by rules in order to simulate their behavior. How to represent most temporal relationships between processes by the use of timers is described in [168].

Another auxiliary mechanism is the *flowing manner* that indicates how an item goes from a store. A flow going into a store will update the store in some manner. The flow from a store may consume or copy the item from the store. Copying is default, whereas a consumption is specified explicitly. In addition to this there has been suggested to annotate PPM-models with performance parameters [244], but these will not be discussed in detail here.

Cross reference between PPM and ONER: Cross-references between the two models can be specified. To do this one indicate the types of data of items traveling along flows and being contained in stores. Items in a flow from a store to a process and also between processes can be linked to a scenario, and thus also implicitly stores and processes.

Process logic description

There are developed two ways of describing the process logic of a process in a PPM, as an I/O-matrix or as a PLD (Process life description).

I/O-matrices The relations between the input flows and the output flows of a process is the i/o condition. This is expressed by a matrix in which each column corresponds to an input and each row corresponds to an output. All rows for one output express the necessary condition for producing the output.

PLD, process life description language: A process life description gives a procedural description of some pattern of behavior. A pattern is attachable to a process or method and can involve interactions with other patterns as well as internal computations. In this way, the description of methods can be applied directly from a PLD being part of a process.

The following elements are used to build a PLD:

- Start: Indicates the beginning of a PLD diagram.
- Receive: Is used to receive data from other PLD models.
- Assignment: Is used for variable-assignment, a PLD block or a subroutine call.
- Choice: Is used to specify selections (if and case-constructs)

- Iteration: Is used to specify loop-constructs (for and while loops).
- Send: Is used to send data to other PLD models.

Figure 7.19 shows a simple PLD model also indicating the link of the PLD to the flows in the process-models.

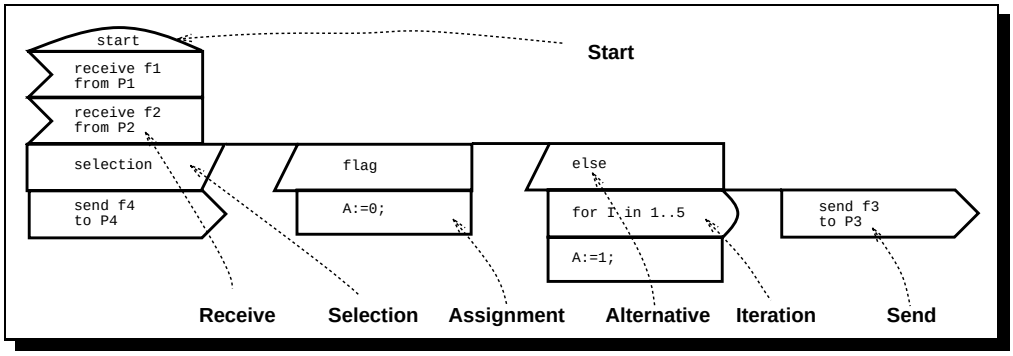

Figure 7.19: Example of a PLD model

Actor modeling language (AM)

An actor represents an executable program or an individual. What is described here is how it is used for describing an applicative actor. The properties of an actor are represented by *functions* and a *working state*. An example is given in in Figure 7.20.

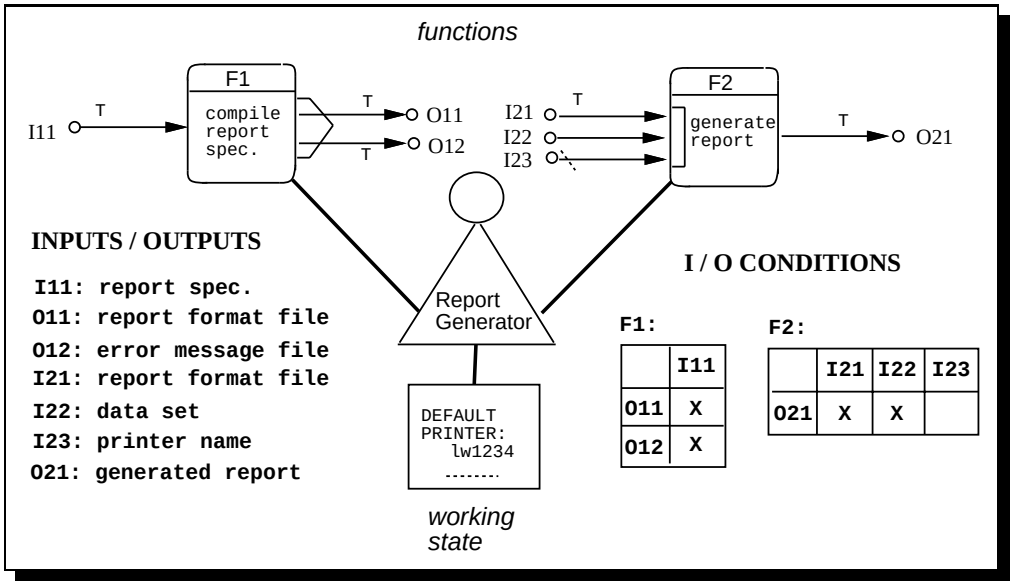

Figure 7.20: Example on an actor model

An actor has one or more functions, each of which may receive some *input(s)* and produce some *output(s)* upon the input(s). The same graphical notation is used for functions as for processes. *I/O conditions* are used to describe the relationships between inputs and outputs of the functions. A *working state* contains data which describe static aspects of the state of the actor.

The mapping between AM and PPM : An actor can be mapped to a reusable process by the following transformation rules:

- The actor itself is mapped into a reusable process.
- The process has an xor input port whose members are the input ports of all the functions of the actor; in the same way, the process has an xor output port.
- The inputs and outputs need to be re-named and, when necessary, some inputs or outputs should be merged.
- the I/O conditions of the functions are also merged with the corresponding changes of names (of inputs and outputs).

Having made the mapping, one can regard the actor as a "general purpose" process. Whenever a function of the actor can support a process, one can replace the process in a PPM by a reference to the general process with the inputs and outputs of the function. According to the mapping rules, the inputs and outputs must be a "compatible" part of the general purpose process.

One can also "group" several processes into an actor. The processes will become the functions of the actor, and the new working state may contain the variables for describing the process logic. After the actor is built one can map it again to a general purpose process, and replace all the grouped processes in the PPMs where they appear by the corresponding references to the reusable process.

UID - user interface description language

Even if the model of the user interface is usually not regarded as a conceptual model, we will briefly describe the languages for developing such models in PPP including the links to the other languages, since the user-interface is an important part of most application systems.

It is separated between the presentation and behavioral parts of a user interface. By the presentation part one means the part of the interface which is visible to the user. By the dynamic part one refer to the interaction between objects of the presentation part, and interaction with other parts of the application. The two parts of UID are called UIP (User Interface Presentation) and UDD (User interface Dialog Description), respectively.

Presentation- UIP : UIP is object oriented.

- Components of a user interface can be modeled as encapsulated objects with a defined set of services and protocols. Execution of an application implies interaction between these objects and other parts of the application by means of message passing.
- Components of a user interface can be grouped in families of objects with similar behavior. UIP make it possible to generate user defined classes of interface components.

Each class in the user-interface hierarchy have a set of properties that can have values, a set of methods that can change or retrieve these values, and a set of events that an object of the class can react to. A state of a screen in a user-interface is defined as a mapping of values to all properties except value contents of all objects in the screen. When one of these changes due to an event, a state transition takes place.

Behavior-UDD: Interaction between components of a user interface and its environment is modeled by UDD. A user interface will be a set of screens which can be thought of as a set of state machines as described above. Transitions are defined by the set of services available in each state and the environment the interface is working in. UDD is based on Statecharts as described in Chapter 5. One Statechart is made for each screen.

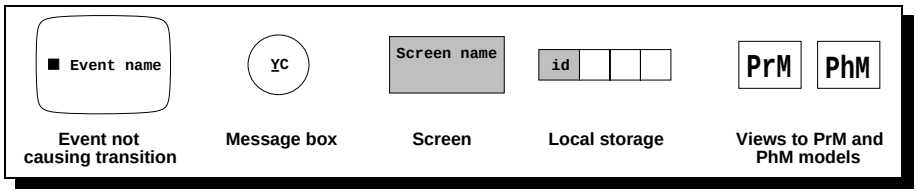

Figure 7.21: Symbols in the extensions to Statecharts used in UDD

The extensions in Figure 7.21 are made to Statecharts in UDD.

- Events not causing transitions: Events which alter data attributes often do not cause state transitions as defined. Such an event is presented by a special symbol within the state.
- Message box: A typical graphical user interface uses message boxes extensively. To simplify the diagram of such user interfaces, explicit message boxes is used in UID.
- Screen: When modeling a screen, one often needs to include other screens in the diagram because of possible transition of control out of the current screen and into another.
- Local Storage Structure: Data structures to store data for the screen which are not visually shown to the user.
- View: Connections to PPM and ONER models.

Often one will need to perform a selection of data from the database when invoking a screen, and this is specified with a scenario including selection criteria (using the defined algebra on ONER models). The PPM view is to indicate the action of an event, thus indicating which data in the user interface (including the local structure) which will be sent to trigger a process. This will be linked to one or more outputs from processes “further down” in the PPM, to indicate when to return control to the user-interface, and where to put outputs.

7.4.2 Methodology

The current development strategy of PPP is illustrated in Figure 7.22.

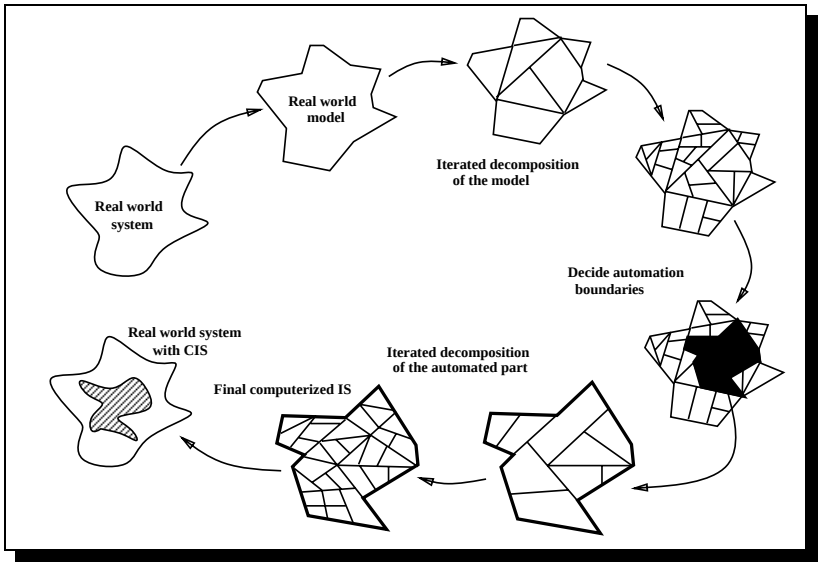

Figure 7.22: CIS-development strategy in PPP

It follows a top-down approach, where specifications are developed in an incremental and iterative manner. The strategy starts out by establishing a conceptual model of the area in question. Details are added during decomposition of various parts in order to formally specify their properties. The development of a CIS is perceived as a mapping between the manual system and a computerized system. A functional analysis of the manual system is thus believed to cover the design of the CIS. Analysis and design are highly integrated. The main issue will be stating which parts of the manual system would be automated by deciding the automation boundary. Having decided on this, the parts to be automated can be further decomposed and specified to a level of formality from which code for the CIS can be automatically generated.

The conceptual models and the user-interface model can be developed independently from each other, and techniques for verification and validation of both parts are supported, both individually and together. We concentrate on conceptual modeling below.

Central to the development of the models is model integration through the use of translations and transformations. Due to the tightly integrated languages, initial models in one language can be generated from models written in other languages. Examples of such transformations and translations are:

- Initial PLD models can be generated from the ports and flows found in PPM-models [335].
- Ports of a process can be abstracted from ports of the process decomposition [340].
- Reports can be generated automatically.

Additional verification support are:

- Prevention and detection of syntactic invalidity.
- Consistency checking: PLD statements are checked for type correctness when they are constructed. For instance, the check ensures that items of a flow that enters a process has the same type as the ones that where sent in the sending process. Logical expressions that are formed by input and output ports are checked for inconsistencies.

Model comprehension is also supported [336]. Three different techniques are partly integrated. A model can be transformed into an *executable program*, a *viewspec* applying similar filtering techniques as in Tempora, and/or an *explanation*. PPP supports execution by providing *translations* from PPP models to various executable target languages such as Ada [116], C [19], Simula [120] enabling simulation also of non-functional requirements, and temporal logic rules in a combination of TEQUEL and C [168, 179, 197, 198] utilizing the Tempora runtime system.

These execution of PPP models all rely on tools outside PPP. In order to have better control over the model execution, an execution approach which offer model execution within the PPP tool is developed [335]. This has many advantages, among which is the ability to perform step-by-step execution, better facilities for model animation, and improved target environment independence. The approach taken has developed a rich, executable, internal language, and provide mappings from PPP models to this internal language.

An explanation generation component that can present various types of model-related data in terms and structures understandable to the users has been constructed, supporting:

- Language explanation: Both semantic and syntactic properties of the languages in PPP are explained.
- Conceptual model explanation: User-tailored explanations of properties of PPP models. Parts of the model may also be included to illustrate the textual explanation.
- Verification checks: Messages reporting on inconsistent or internally incomplete models are presented as texts explaining why the model is illegal.
- Model execution: During model execution, explanations justifying the results or describing the user interaction can be generated.

The explanations are generated using a number of strategies. Each of these corresponds to a specific explanation type supported by the system, and describes how the type is realized using other strategies and/or including various types of model-related data. Combining a series of such strategies, a hierarchical explanation structure is constructed, in which non-leaf nodes are names of strategies used and leaf nodes refer to the modeling language, the conceptual model, or the trace from executing the model. As opposed to traditional paraphrasing systems, this component integrates different sources of data, and both the structure and content of the explanations generated may be user-tailored [114]. The user model is divided into two submodels:

- The user characterization submodels, which characterizes the user in terms of which elements of the source model are accessible to him.
- The user knowledge sub-model, which specifies the parts of the source model that are known to the user. The source model contains both the conceptual model and the language model describing the conceptual modeling languages used.

Also, views from the conceptual model can be requested from the abstraction module discussed earlier and included as illustrations in these explanations.

A PPP model is stored in the repository according to a defined storage schema for (1) the model structure, (2) the model content, and (3) the model layout. A description of the repository solution is given in [341]. The architecture is made to support sharing and distributability of results. Additional support for cooperative work in the development of conceptual models is given in [4] including advanced versioning mechanisms.

7.4.3 Evaluation

Classification of PPP

- Weltanschauung: Objectivistic.
- Coverage in process: Focused on development.
- Coverage in product: A single application system, but have possibility of modeling organizational-wide models.
- Reuse: Generative reuse through the use of already developed models in later phases and code-generation. Compositional reuse through actors and reusable processes.
- Conceptual modeling: Methodology based in a large degree on active use of formalizable conceptual models.

Language quality

- Domain appropriateness:
  The structural, functional, and behavioral perspectives are supported well. Although not supporting traditional state-diagrams or Petri-nets, triggering makes it possible to model the same dynamic properties as is illustrated in Figure 5.10 [168]. Statechart-modeling is used in the UID. Organizational modeling, and speech act modeling are not supported. Although it is possible to model single actors, actor-interaction is poorly supported. Also some limited support of the rule-perspective is given, but this is basically to model constraints on the structural model. Whereas some parts of the object-oriented perspective is supported, several more specific object-oriented features are not supported such as advanced inheritance mechanisms, detailed method-specification, modeling of entities, and state-transition diagrams for entities. Some temporal expressiveness exist through the use of timers. An ad-

ditional feature found in PPP is the possibilities of integrated modeling of non-functional aspects such as performance and the user-interface.
The languages have both general and some very specific constructs, and are composable in a similar way as traditional data, and process modeling languages, which means that the composibility is somewhat restricted. The languages can be flexible in precision.

- Participant knowledge appropriateness:
  PPP is based on much-used modeling languages such as ER and DFD, although extended to increase the expressiveness. For those familiar with this basis, the PPP languages as such should be rather simple to learn and understand. The indication of cardinality uses letters (p/f) which are language specific. This is a minor problem though, and could alternatively be supported in a tool for modeling by changing the external representation according to language. Modeling of inconsistencies is not supported.
- Participant interpretation enhancement:
  The difference conceptually between function, process, and method is unclear. When it comes to function and process, these are also represented symbolically in the same way. The external agent-symbol is used both for actors and for roles, i.e. the use of this phenomena is not uniform. On the other hand, the same symbol is also used for actors, which can be described by their functions and their state, whereas an external agent can be described by rules.
  As indicated in Chapter 5.2.2, the flow-symbol in PPM is overloaded. A flow might indicate a transportation in space of some item, or only indicate a link between two model elements usually depicting a temporal relationship.
  The indication of "cardinality" of types uses different representation of optional, i.e. that a set type is optional is indicated by a number (or nothing), whereas a composite type is indicated using an asterix.
  Processes with large port-structures easily become complex, but this is a drawback of the increased expressiveness. Similarly as in Tempora, the specification of ports might result in more crossing lines, and longer lines than would else be necessary.
  The use of emphasis by solidity is obscure, emphasizing sinks in process models and methods with side-effects in the data models.
  Visual abstraction mechanisms for the ONER language is lacking, neither having the possibility to represent classes in a compact manner like in OMT, or being able to use grouping, like in Tempora. On the other hand, some specific clusters of classes can be utilized to improve the situation.
  The 'start'-symbol in PLD is empty.
- Technical actor interpretation enhancement:
  The modeling languages have a defined syntax and operational semantics which can be used for the creation of prototypes and explanations. Little has been done to improve executability in the approach for efficient execution.

Potential for creating models of high quality

Tools have earlier been made to support parts of the approach. One are currently in the process of implementing a new set of tools that are supposed to include most of the above described functionality.

- Physical quality: Repository and versioning mechanisms are described and are currently being developed to support a tool for geographically dispersed cooperating system devel-

opers. The versioning mechanisms are created specifically for the problem of versioning large conceptual models, and include novel features in this area.

- Syntactic quality: The modeling languages are formally defined and thus opens up for both error prevention and error detection. This is also to be supported in the new tool.
- Semantic quality: Tool for updating models has been developed. Algorithms for advanced consistency-checking of both the process and data-model have been devised and is to be incorporated in the tools.
- Pragmatic quality: Several different techniques for potentially increasing the pragmatic quality of models have been described, and are partly integrated, such as user specific explanation, execution, and filtering mechanisms. Also primitive simulation can be performed. On the other hand the mechanisms are only partly integrated, thus even if the separate approaches have been shown to be among the state of the art within the separate fields, more work must be performed to be able to use the techniques effectively in concert.
- Social quality: Not explicitly addressed, although being discussed briefly in [4, 282].
- Knowledge quality: Not discussed in any depth.

7.5 Chapter Summary

We have in this chapter described and briefly evaluated several existing methodologies which are applying conceptual modeling extensively. The evaluated approaches were: OMT, Tempora, SAMPO, and PPP.

As the evaluations have indicated, they all have different weaknesses and areas for improvement, both when it comes to the languages used, and the techniques for supporting the creation of conceptual models of high quality. The methodology definitions differ a lot in the amount of details that are described.

In the rest of this thesis, we will use the PPP-languages as an outset for further work, extended in areas which was looked upon as mandatory, i.e. modeling of actors, roles, rules, and speech-acts are supported in addition to the aspects already covered in PPP. The choice of building upon PPP is partly pragmatic, since we have a long time experience with this framework, and also are developing tools supporting main parts of this approach which can be extended with our suggestions for language extensions at a future point in time. The primary assessment of PPP using the quality framework also indicates that although certain deficiencies can be identified, we believe that it has a potential, with the right tool and method support, to be used to create conceptual models of high quality.

Chapter 8

Methodological Framework

We will in this chapter describe our suggested methodological framework for CIS support. We have coined the framework "system devtenance" to indicate the removal of the difference between development and maintenance. The description in this chapter will be on a high level of abstraction, without linking to a specific set of conceptual modeling languages. This will first be done in Chapter 10. The overall goal of the methodology is to give adequate CIS-support in the organization, as judged on system quality, data quality, use, user satisfaction, individual impact, and organizational impact. At the same time the resources used for this should be adequate, which among other things in our view necessitates a low percentage of functional maintenance.

The framework is presented on the portfolio and application systems level in Section 8.1 and Section 8.2 respectively. A discussion on stakeholder participation in systems devtenance is given in Section 8.3.

8.1 System Devtenance

The framework is based on constructivism, and includes both development, use, and maintenance of the whole application systems portfolio by emphasizing on conceptual modeling and reuse. We have chosen to present it on the portfolio-level first, since this is an important aspect of the methodology, and since it is not so that a transition on the portfolio-level necessarily only involve one application system.

Figure 8.1 illustrates the overall framework: The information system of the organization contains at any time a set of application systems. This set is termed the application systems portfolio or portfolio for short. We look upon the changes to this portfolio as a transition from one state of the COIS to another. It is also possible to perform transitions in the organizational information system by for instance changing some manual procedures without changing any part of the portfolio, but we will not discuss these kind of changes here.

Transitions are divided into two types, **visible** and **buffered**. Visible transitions are those being performed as part of the daily routines of keeping the COIS operative, and include emergency application system fixes. This work is performed by a front-end organization being part of the COIS-department. All other changes are performed in projects, in the sense that they have a more or less defined scope and the time-span of the endeavour is limited. In addition will the results from the project be buffered from the daily work of the organization until the project is finished, whereby the changes are committed to (parts of) the organization. Projects involve persons from the back-end of the CIS-department. In this way, it should be possible for the back-end to concen-

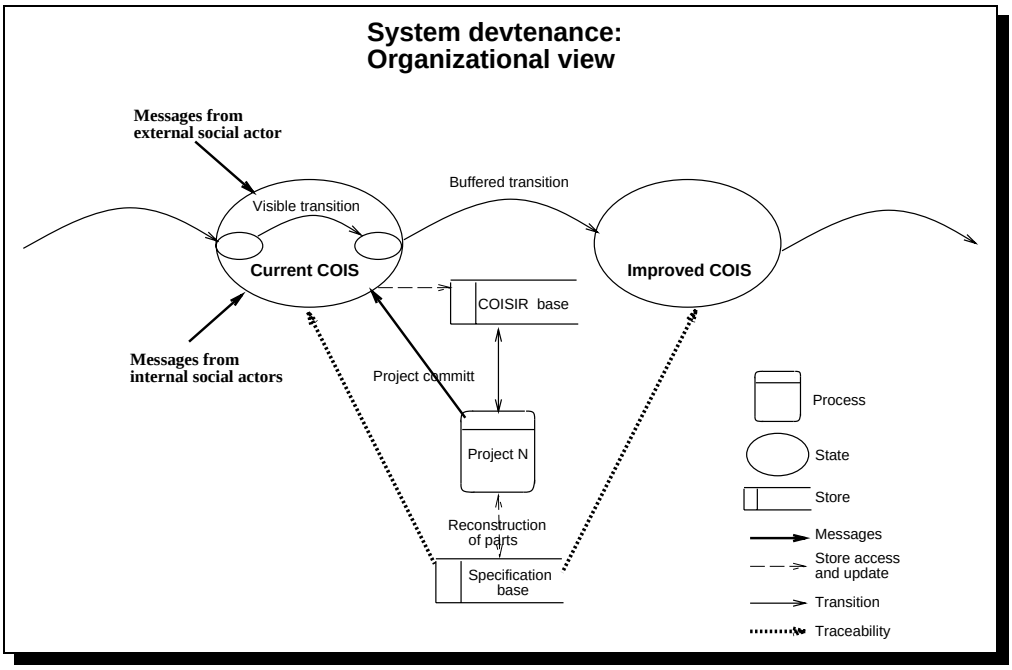

Figure 8.1: A framework for system devtenance

trate on more long-range devtenance projects, and be buffered from the daily request from users and managers, and thus controlling the problem of changing priorities described in Chapter 3.

Continuously, external and internal forces pressure for transition of the COIS. All COISIRs [1] from the different stakeholders of the OIS are logged and assessed together with other demands for changes and stored in the COISIR base. Changes are either adaptive through the conscious development of an information system plan, or natural due to factors beyond the control of the organization.

When the discrepancy between the contents of the COISIR base and the current CIS-support are found unacceptable according to some measure, a project can be established to address the discrepancy. Also more pro-active project establishment methods might be used. Project establishment should be a participative process, involving stakeholders of the area that is being addressed.

At the outset of a project, a specification base being related to the CIS-support of the organization is supposed to be present, and will be accessed and updated during a project by the use of a (set of) internal supportive actors (e.g. CASE-tools). When the project comes to an end, the changes are committed to the COIS and the traceability between the specification base and the COIS is kept up when the COIS is promoted to a new stable state. All through development, the version of the specification of the current CIS is retained so that it is always possible to reestablish the previous mapping if necessary. The COISIR base is updated continuously, marking the changed status of investigation reports and the discrepancy between the perceived need for CIS-support and the current situation.

Whereas Figure 8.1 shows the simplified situation with one project being performed at the time, it is obviously possible to have several parallel projects updating potentially overlapping parts of the specification base. A situation parallel to this is when a visible state change is per-

[1] Computerized Organizational Information System Investigation Report.

formed at the same time as a devtenance project is underway.

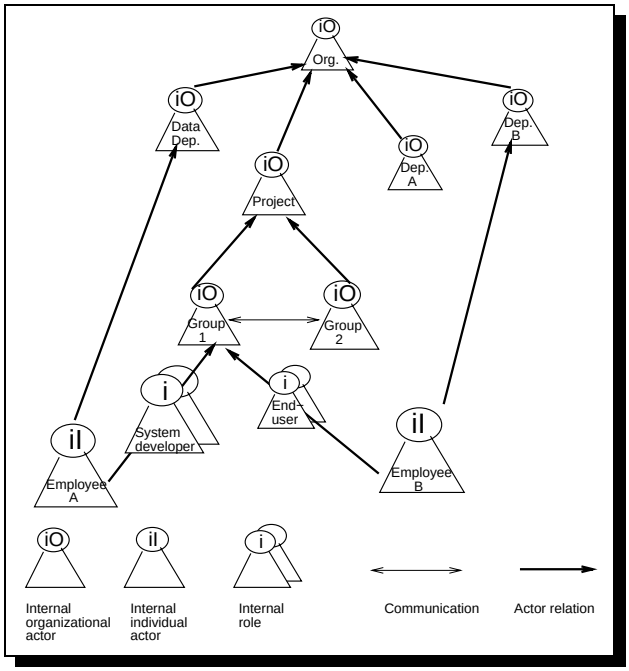

Figure 8.2: Actor model describing a project organization

Another simplification is the two-level structure as pictured in Figure 8.1. Figure 8.2 depicts a situation with an organization having two department in addition to the data department, which have set up a project containing two groups. Group "1" consist of among others an employee "A" that is part of the data department which are part of the group in the role of a system developer, and an employee "B" of department "B", who is part of the group in the role of end-user [2].

The different transactions levels in this case are illustrated in Figure 8.3. Within the projects, typically all developers, also employee "A", have their private workspace where changes can be performed. When these are completed, they are committed to a higher level, being visible for more people involved with the project, here group "1". Several levels within the project might be perceived, in the example only one additional level is pictured. The next level is to commit the changes to division "B" of the organization which might test out the changes, before the changes are committed to the whole organization. It is also possible that one wants to rollback some changes, going back to a previous state. Also less stringent sharing-mechanisms than those given in a pure nested transaction model should be available, for instance publish/subscribe [4] illustrated in Figure 8.2 by the interaction of group "A" and group "B". We will not go into detailed discussions on transaction mechanisms and configuration management in this thesis.

8.2 Application System Life Cycle

The life-cycle of a separate application system is illustrated in Figure 8.4. There are some similarities between this life-cycle and the STEPS and Tempora-methodologies described in Chapter 4.2.11 and Chapter 7.2 respectively.

[2] The actor-language used in the model will be further described in Chapter 9.

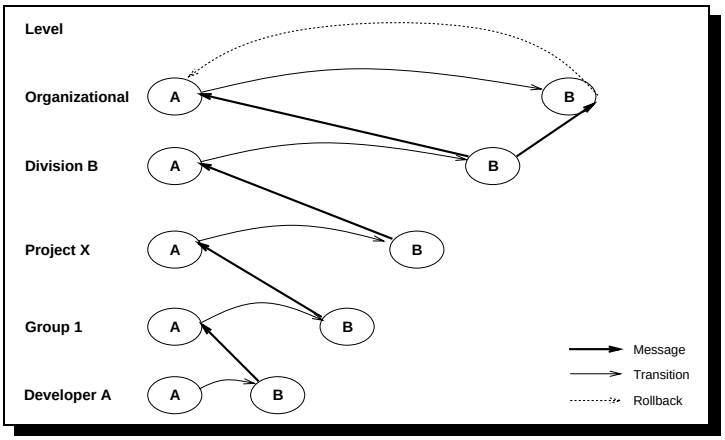

Figure 8.3: Nested transactions in system devtenance

To illustrate several kinds of projects, we might first imagine that there is perceived a need of supporting areas applying technical actors which were not previously supported in this way.

A general project is divided into four "phases" based on the use of conceptual modeling:

- Development of conceptual models of the perceived current situation.
- Development of conceptual models of an perceived improved situation. This is not supposed to take computer specific aspects into account.
- Development of conceptual model of the CIS.
- Implementation. Based on the selected supportive actors, this is a more or less automatic translation of the design into the application system.

Proposing a set of phases does *not* mean that an approach based on the waterfall methodology with disjoint phases performed in a strict sequence is promoted. Different project models can be overlaid the proposed approach in order to meet the management requirements of a development project. The choice of this will be based on different factors as discussed in Chapter 4.

Conceptual modeling is to be a participative process, taking as an outset the project description created in the project establishment, the parts of the COISIR base which are relevant to the area together with the local reality of the participating stakeholders. In addition to this, knowledge about existing software actors, and models of these can be taken into account.

The basic principle is to develop a set of conceptual models on each area above to a point where:

- There is feasible agreement among the participants that the model will address the perceived problems.
- A common understanding of the consequences of the proposed model has been developed. The common understanding is to be achieved by several means. All participants must have a good understanding of the models and modeling languages being used. It is not possible to avoid that the developers will know the modeling languages and thus also models made by using these languages better than most users, so one must acknowledge the danger of model monopoly (see the next section). Model monopoly can also appear in situations where one group of the stakeholders are able to state their views using languages and models which are difficult for other groups of stakeholders to comprehend. The models can be validated through transforming them to prototypes which then can be run, explained and reasoned about in addition to observing the models as is and by using filtering and presentation tech-

niques based on aesthetics as discussed in Chapter 6.

- The model is feasibly complete and ready to be taken over to the next step, or an application framework which match the developed model sufficiently well, and which one can use in the rest of the project, is identified. The application framework can be either an external or internal software actor. The descriptions of external application frameworks are found in an application and component base which are separate from the specification base of the COIS. Often such application frameworks will not be available, or the mismatch between the participativly developed models and what is supported in existing frameworks is too large for it to be applicable.

  In other cases, a strategic decision on main supportive software is made in the project establishment. An application framework can also be used at the outset of the project, representing a certain view of the problem to be solved which can be compared with the models developed independently by the project participants.

In addition to storing the decided requirements, also the variety of views and the relevant parts of the process of coming up with the current model, are to be stored in the specification-base and should be available when later versions of the application system are to be developed.

The implementation can either be a direct translation of the specification and design, including additional design decisions necessary to enable code-generation as parameters, if the CASE-tool provide the necessary translation between the detailed conceptual models of the specification base and the supporting software of the application system. As indicated in Chapter 3, total code-generation is supported from some tools to some supporting actors. See also e.g. [88] for a description of practical application of this kind of technology. If this functionality is not available, a more traditional implementation phase will be called for, potentially applying frameworks generated from the conceptual models. This task is indicated to be primarily performed by the systems developers, but in the case of a more traditional detailed design and implementation, this must be followed by user acceptance testing as used in more traditional methodologies. Also changes done to the models due to formalization must be re-validated.

When the application system is committed to (parts of) the organization, the request base is updated accordingly. When using the application system on a regular basis, the users will experience breakdowns in the phenomenological sense. Facilities for easily being able to report this so that it can be added to the COISIR-base should be available. In addition should the relevant conceptual descriptions of the application system be available to the end-users to improve on the internalization of the application system.

The front-end of the CIS-department will perform the necessary corrective maintenance on the system, and receive the COISIRs from the users, performing a preliminary assessment of them. Also other user support tasks will be performed by this group on a daily basis. Their work can influence the specification base and will also influence the COISIR base as errors are fixed and new potential needs are discovered.

When it is decided to make a new version of the system, a new project is established, and much of the same will take place as above. Some differences are:

- The users (and in this case, all users) have had the opportunity to have more hands on, realistic experience with the application system than it is possible to achieve in a development project. Thus, the users will have a much better picture of what the problems and possibilities of the application system are and what the conceptual models of the systems mean in relation to their own working situation. With this background, it should be easier for the users to come up with constructive suggestions for improvement.
- If the application system has been developed by using conceptual modeling, there exist al-

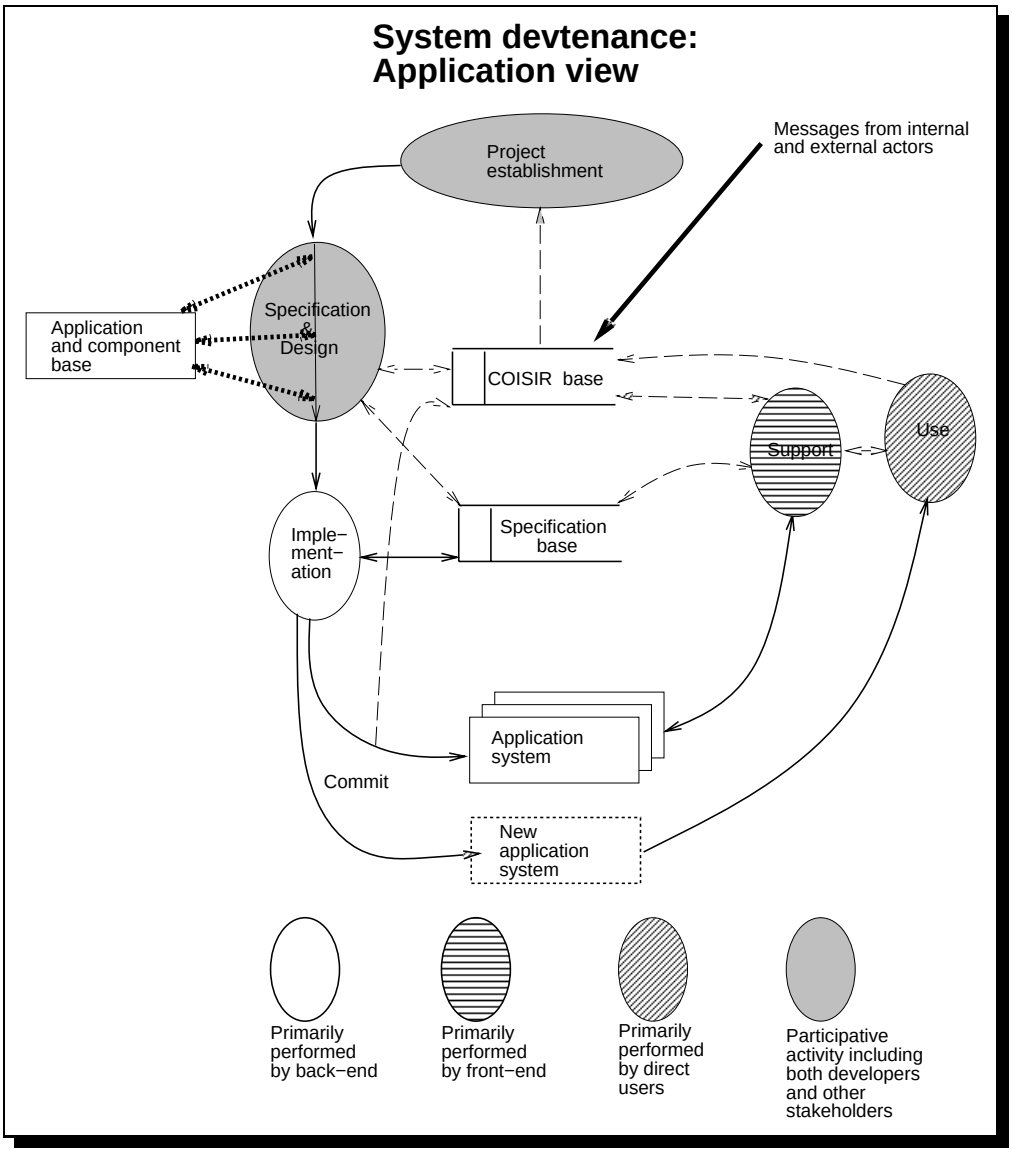

Figure 8.4: System devtenance life-cycle

ready a conceptual model of part of the current organizational reality (i.e the existing application system) as an outset for further development. It is important that this is only looked upon as one of many possible CIS-solutions for the problem area, not prematurely limiting the perceived solution space. By retaining the defeated views in the original development, we believe it will be easier to avoid premature closure than if one did not retain this variety.

Also other scenarios can be perceived. One is that it is no longer desirable to maintain the existing application system, for instance if it should be moved to another platform because of integration of different application systems. Then one should be able to reuse the old specification in a much larger degree when they are represented as conceptual models being independent of the implementation of the system. One can then update this and transform this/re-implement it in a new application system. Even larger savings can be imagined when all the application systems that one wants to integrate are described using conceptual models. In this case, matching technique similarly to those used to assess the applicability of application frameworks can be used between

the potentially overlapping conceptual models of the two application systems to be integrated. We will return to further details on this in Chapter 10.

8.3 Stakeholder Participation in System Devtenance

Applying social construction theory implies a methodology with stakeholder participation in focus, since it is not possible for a set of outsiders to come up with objectively true requirements to the application systems. Many of the requirements to the system do not exist a priori, but are constructed in the process of application system devtenance.

We will in this section first briefly discuss different aspect of participation, before outlining a way of having user participation in systems devtenance.

8.3.1 Background on Participation

A **user** of a *CIS* is defined as a person who potentially increases his *knowledge* about some *phenomena* other than the *CIS* with the help of the *CIS*. An **end-user** increases his *knowledge* in areas which are *relevant* to him independently of the actual *CIS* by *interacting* with the *CIS*. **Indirect users** increase their *knowledge* by getting results from the *CIS* without *interacting* directly with the *CIS*.

The above is somewhat different from how 'user' is often defined, terming the system development and maintenance personell as 'primary users' [133] or technical users. Not including these as users in the following discussion do *not* mean that they are not important stakeholders of the CIS support.

There exists different forms of participation:

- **Direct participation:** Every stakeholder has an opportunity to participate.
- **Indirect participation:** Every stakeholder participate more or less through representatives that are supposed to look after their interests. The representatives can either be selected by e.g. management or elected among their co-workers:

According to Heller [128], participation is sharing power and influence. He has divided the degree of influence and power into six categories as illustrated in Figure 8.5.

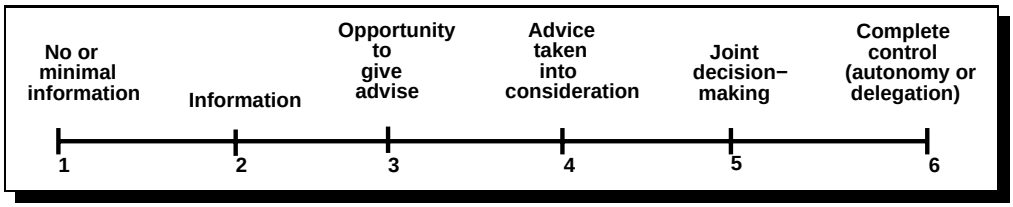

Figure 8.5: Scale of influence and power in participation (From [128])

When we discuss user participation in this thesis, we will mostly refer to categories 4 and 5 on this scale.

User-participation can be assured by the use of different links between the users and the system developers, some of which are traditionally used primarily under customized development (C), and some primarily under packaged development (P) [162].

Due to the large number of potential stakeholders in most devtenance projects, in most cases representative participation will be the only practical possibility. From the point of view of social

Link	Used in P/C
Facilitated team (e.g. JAD [12])	C
MIS intermediary	C
Support line	P/C
Survey	P/C
UI-prototype	P/C
Functional prototyping	P/C
Interview	P/C
System testing	P/C
Email/bulletin board	P/C
Usability lab	P/C
Observational study	P/C
Marketing and sales	P
User group	P
Trade show	P
Focus group	P

Table 8.1: Links between users and developers in CIS-support

construction, it is doubtful that a user representative can truly represent anyone else than himself. On the other hand, even if the internal reality of each individual will always differ to a certain degree, the explicit knowledge concerning a constrained area might be more or less equal, especially within groups of social actors [102, 248]. When it comes to suggesting improvements of the current information system of the organization, direct participation should be possible. Also in the project establishment, a larger proportion of the stakeholders should be able to participate. To ensure widespread participation this means that the methodology should support:

- An efficient system for the issuing and follow-up of COISIRs. Efficient follow-up does not mean that it should result in immediate changes to the portfolio.
- A possibility of using conceptual models of the CIS in the sense-making process also during use of the system, not only during development and maintenance projects.
- The possibility to change an existing application system easily.

A main technique to achieve this is to have short intervals between releases, thus setting up for a stage-wise development methodology during traditional development projects so that the end-users can come up with their views more rapidly to enable improvements. This can also have the positive effect of fast payback of project-costs.

Many arguments for having participation have been given in the literature see e.g. [110, 231] for classifications of arguments. In this thesis, stakeholder participation is basically motivated through a cost-benefit-perspective on the long run. Since all stakeholders have their individual local reality, everyone have a potential useful view of how the current situation can be improved. Including more people in the process will ideally increase the possibility of keeping up with the ever more rapidly changing environment of the organization. Added to this is the general argument of including those who is inter-subjectively believed to have relevant knowledge in the area, and which are perceived to be influenced by the solution. As indicated in Chapter 3, stakeholder participation appears to be a general indicator for (development) project success as perceived by all the different stakeholders.

8.3.2 Principles of Stakeholder Participation

Participation in system devtenance projects should take place according to Figure 8.6. The figure is adapted from a similar figure used in work in organizational development on participative action research [82] where a similar philosophical outlook is used.

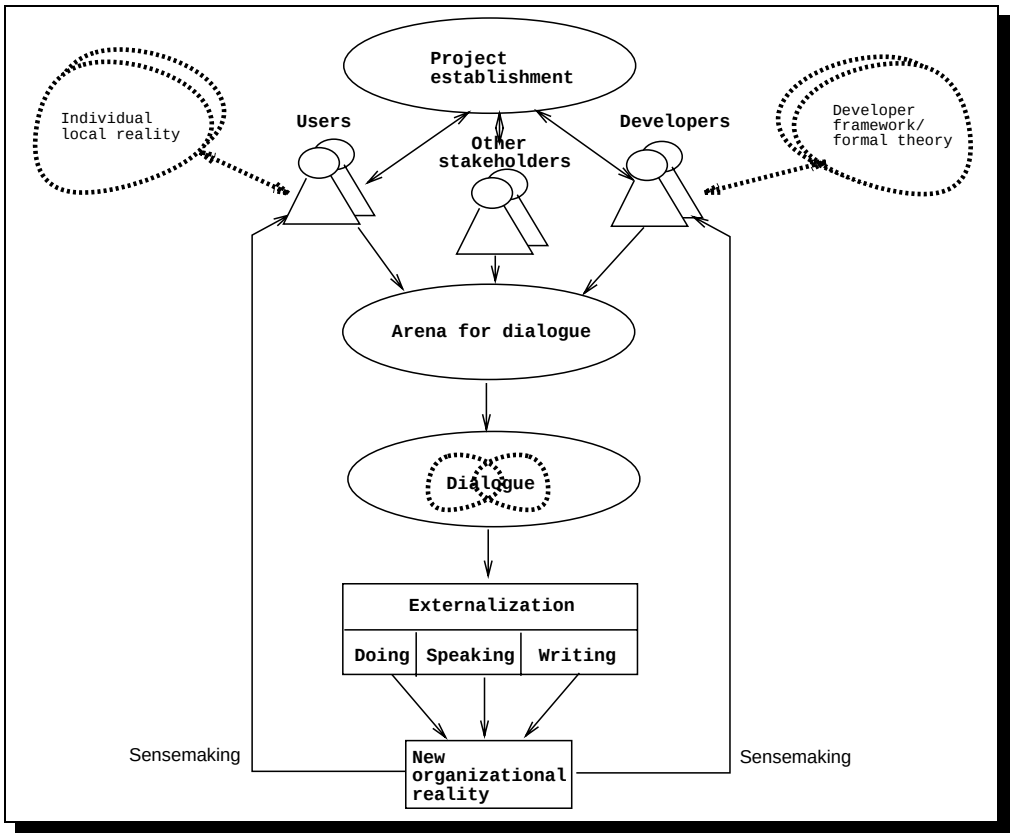

Figure 8.6: Co-generative learning in systems devtenance

Before the outset of the project, a project establishment phase has been performed on some level and are taken as the starting point for the devtenance project. Users and other stakeholders all having their own internal reality and developers which also have individualized theories on the organizational reality and systems devtenance meets on arenas for dialogue.

The developers are more or less to be regarded as outsiders to the user community, either they are external consultants, or they are working in the CIS-department of the organization. Both the outsider's (i.e systems developers) local reality and the insiders (i.e. primarily users) local reality are regarded to be equally valid. This is necessary to avoid the problem of model monopoly in user participation where the system developers' or some group's perception of the world dominates the discussion, something that can result in what Habermas [118] terms 'naive consensus'. Bråten has developed the following guidelines [34] to avoid this so-called model monopoly:

- 1. Participants must be aware of the mechanisms of model monopoly.
- 2. One must have several different models available, and accept the validity of the individual models of each participant.
- 3. One must use time to develop an understanding of the participants own premises and models.

- 4. One must have an atmosphere that facilitates a democratic dialogue to eventually reach a common understanding of the problem, or at least an understanding of what the disagreement is all about.

Stated in our terminology, the last point means that one needs to achieve feasible comprehension and agreement on the constructed conceptual models.

In addition, it is important that all the participants are given training in the use of the conceptual modeling languages that are used. As stated by Heller [128], true participation necessitate competence. If the user participants are supposed to be actively taking part in the model construction, they must have developed necessary skills in this in addition to having skills related to the application domain. This means that just being shown existing models that have been made by others for "validation" will not be sufficient, even when they are accompanied by intelligent explanations.

As mentioned by Bråten the dialogue between the stakeholders should ideally follow Habermas' rules of a democratic dialogue [117][3].

- The dialogue is a process of exchange: points and arguments move to and fro between the participants.
- All stakeholders must have a possibility to participate. *As already discussed, in our case not all stakeholders are able to participate in many projects directly. On the other hand, the user representatives who participate should be given the possibility to participate fully. When the results from the projects are committed to the organization, all users are able to participate by issuing COISIRs, and being part of e.g. search conference groups deciding on additional changes, and thus influencing the future organizational reality.*
- Possibilities for participation are not enough: Everybody should also be active in the discourse. *It is difficult to assure that people are active. One factor here is that one should enable them to be active, by giving them sufficient training in the use of the modeling languages and about computer technology in general. Another is that they are given sufficient time to participate.*
- As a point of departure, all participants are equal. *This is not to disguise the fact that the participants have different skills and knowledge. What it states is that no view of any participant is implicitly better than any others before being discussed. Note that this is similar to the second guideline of Bråten.*
- Work experience is the foundation for participation. *This can be re-phrased with that it is the individuals local reality which have been achieved partly through work that is the foundation for participation.*
- At least some of the experience which each participant has when he or she enters the dialogue must be considered legitimate.
- It must be possible for everybody to develop an understanding of the issue at stake. *This means in our terminology that the sense-making process must be supported, and one must avoid model-monopoly situation.*
- All arguments which pertain to issues under discussion are - as a point of departure - legitimate.
- The dialogue must continuously produce agreements which can provide a platform for investigations and practical action. *This is similar to the fourth point of Bråten, even if we modify this requirement to feasible agreement.*

In this connection, it is also enlightening to mention Rittel's suggestions for dealing with wicked

[3] Our comments are written in *italic*.

problems as discussed in Chapter 2.2 [268]:

- The expertise needed for dealing with wicked problem is distributed among several people. Those people who are the best experts with the best knowledge, are usually those who are likely to be affected by the solution, i.e. the different stakeholders.
- Nobody wants to be planned at.
- One shall have a "transparent" solution process. There is always 'ought-to-be' statements involved in the process. Thus if you look only at the outcome of a process, you cannot reconstruct which deontic statements have entered into the argument leading to the solution.
- Everybody is entitled to exert his judgement about the solution. We need procedures which enable us to explain to each other why we think as we do.
- The analyst is a "midwife" for problems, not a doctor offering solutions.
- There is a need for dialogue between the participants to create mutual understanding and mutual learning. Wicked problem-solving must be understood as an argumentative process: One of raising questions and issues towards which you can assume different positions, with the evidence gathered and arguments built for and against the different positions. The various positions are discussed, and after a decision is taken one proceeds until the next question arises within the process.

The participants will as a result of the democratic dialogues on given arenas externalize their reality in different ways. The externalization may result in both linguistic and technological artifacts, being first visible only within the project or a limited group in the project. The linguistic artifacts will partly be in the form of conceptual models, which are one of the ways of externalizing ones local or shared reality through writing. Other linguistic artifacts can be made using natural language. Explaining these to others and arguing about them will be externalization in speech whereas when creating a prototype based on the conceptual model other can use this for externalization of additional views through action. The models will be externalized and first be part of a larger organizational reality when committed to the organization in the form of an application system produced on the basis of the models.

From these externalizations the different stakeholders will be able internalize the changes in their own local reality, and thus be able to discuss discrepancies and ultimately be able to apply the CIS effectively.

8.4 Chapter Summary

We have in this chapter given a high-level presentation of our suggested methodological framework coined systems devtenance. The framework is based on constructivism, and includes both development, use, and maintenance of the whole application systems portfolio by emphasizing on conceptual modeling and reuse.

The framework has been presented on both a portfolio and an application level, and the principles for user participation within the framework is outlined.

In the next chapter, we will look on one way of extending the modeling languages of PPP as presented in Chapter 7.4 to be more generally applicable, before a way of using the complete conceptual framework within the methodological framework presented in this chapter is given in Chapter 10.

Chapter 9

Conceptual Framework

The conceptual framework which will be used in the rest of the thesis is an extension of PPP as presented in Chapter 7.4. In this chapter only the extensions will be presented, for a presentation of the existing PPP-languages, we refer back to Chapter 7.4.

We adopt ONER, PPM, PLD, and the UID of the PPP approach more or less unchanged. Even if the evaluation indicated room for improvement of the existing languages, we will not concentrate on addressing these, but rather on addressing the perspectives that are currently poorly covered in PPP, i.e. rules, actors, roles, and speech acts in a way that can support a methodology covering the development and maintenance of portfolios of application systems applying reuse and being based on social construction theory. This extension *will* imply some changes to the PPM-language.

The framework is extended in two ways:

- The actor modeling language will be extended to be able to describe both social and technical actors, their interaction, and roles. It will be shown how this extension can be used for organizational modeling, portfolio modeling, extensions of process modeling, and speech act modeling.
- Tempora ERL will be integrated with the other modeling languages as outlined in [282]. This rule language will in addition be extended with inter and intra-rule deontic operators, the last as part of supporting rule-hierarchies. It is also indicated how the modeling of speech acts and deontic rules might be integrated.

In doing the extensions, we have tried to take into account the discussion of language quality as discussed in Chapter 6. In particular we have tried to not introduce too many new major phenomena classes, making the number of phenomena too large too handle, but instead try to improve expressiveness mainly by integrating the various languages further.

After presenting the language extensions in Section 9.1 and Section 9.2, we will indicate how the existing quality-enhancing techniques in PPP can be extended to take the conceptual extensions into account. In the main report we have discussed necessary extensions of the meta-model, prototype generation, and consistency checking. In addition do we introduce the use of the driving questions technique. Additional details on integration regarding general execution, explanation generation, filtering, and tool support are presented in Appendix D. A comparison with related work will be given as part of the overall evaluation in Chapter 11.

9.1 Extended Actor Modeling Language

We will first give the motivation for extending the actor modeling language of PPP:

- It should be possible to support the modeling of a CIS-portfolio, not only a single system. Since all application systems can be described as actors, and since we find it important to be able to have an overview of the whole COIS, it should be possible to create overviews of the interaction between technical actor.
- The dual use of external agents and actors in the existing PPM is problematic, since it appears that we use the same symbol for different phenomena. On the other hand, external agents are not used consistently in a PPM, sometimes being used to symbolize a role (e.g. clerk/customer), and sometimes to represent an individual actor, for instance an application system or organization.
- An organizational model being a combination of roles, individuals, and organizations can not be properly modeled in a semantic data modeling language like the one that is used in PPP, since this is class-based. The need for modeling also on the instance level is generally acknowledged [146]. Also based upon our philosophical outlook presented in Chapter 1, we find it important to be able to represent individuals more properly, and not only classes.
- Traditional process modeling languages, which the PPM language is an example of, do not give us any good way of modeling the human actors and the interaction among those. The actors are only represented on the outside of the system, and not as part of the system. This critique is also raised in [115] where the need for the modeling of responsibilities and goals of the actors is argued for.
- The explanation generation of PPP applies an underlying user-model. It would be beneficial to link this to a more general actor model, instead of having a separate treatment of these. One could also extent this to support user-modeling also in the application systems.
- It would also be possible to link some of the human actors in the model to a project management tool. This area will not be pursued in full in this thesis.

In addition should the language retain the possibility of the existing actor modeling language in regard to the description of singular technical actors to support reuse.

9.1.1 Modeling Language

The actor modeling language is based on work within the actors and role, object, and communication perspectives as surveyed in Chapter 5 and Chapter 7. It has also been influenced by the current PPP, and experiences through case-studies using intermediate versions of the language. The main symbols used for the modeling of actors are shown in Figure 9.1.

An actor is represented with the same symbol as earlier (cf. Chapter 7.4), whereas a role which an actor can fill is given a special, although similar symbol. An *agent* is depicted as a combination of these symbols, but this symbol will only be used in special cases.

Actors and roles can be connected in the following ways:

- Relationships: Four general relationships are defined:
  - An actor can be *part of* another (organizational) actor. If actor A is part of actor B, then the set of persons or systems that is contained in actor A is a subset of the set of persons or systems that is contained in actor B. An actor can be part of many actors at the same time.
  - An actor can (be expected to) fill a role. One actor can fill more than one role, and a role can be filled by more than one actor. A role can be identified as a set of expec-

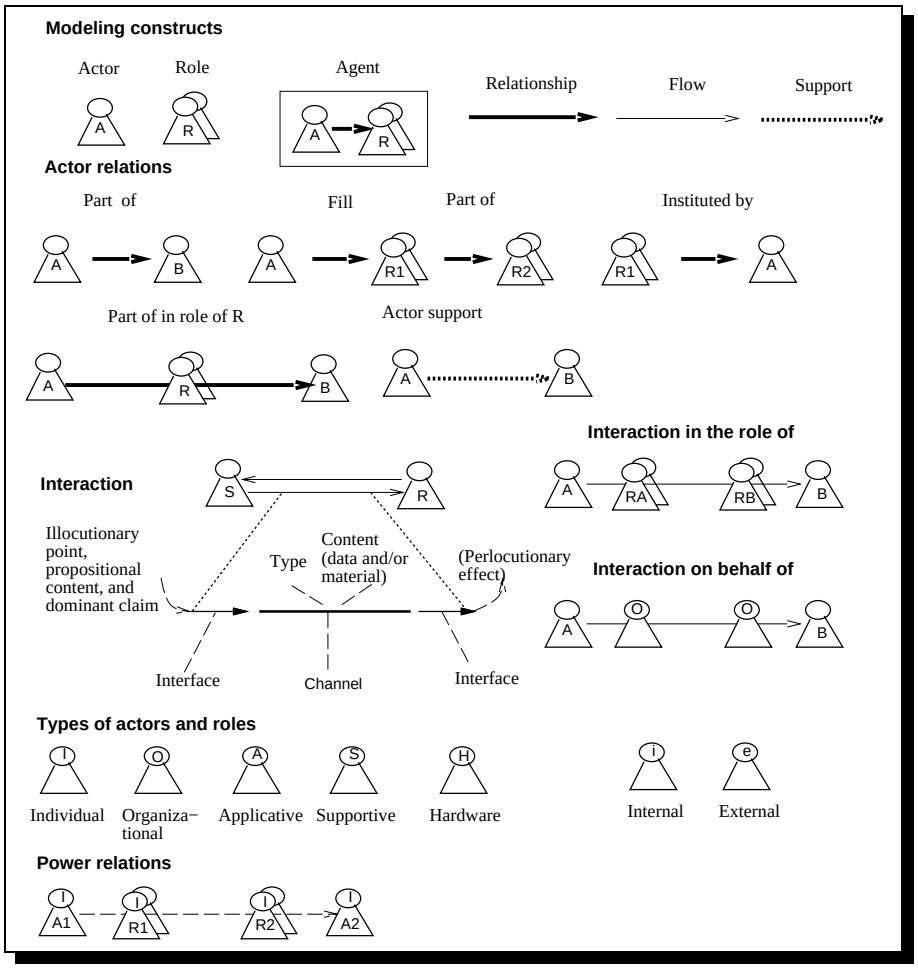

Figure 9.1: Basis for the actor modeling language

tations which potentially also will apply to the actors that fill the role at the time. A role can thus also temporarily be identified with the set of actors that fills it in a member/set relationship. We will return to mechanisms for inheritance between roles and actors throughout the chapter.

  - A role can be *part of* another role, i.e. some of the expectations of a role also applies to another role. This relationship will often directed the opposite way of how it is directed between the structural aspects of the same roles in an ONER model.
  - A role is instituted by one or more actors. The set of expectations to a role often comes from one or more actor. If there are specific expectations to an agent which are important to represent, this can be depicted using the agent-symbol.
- Support: Actors and roles can be said to support other actor and roles in performing their tasks. We will indicate what this means in more detail when discussing types of actors and roles below. It is also possible to indicate that actors *potentially* support other actors.
- Communicate: Actors and roles can communicate by exchanging items over flows. Strictly speaking, roles can neither support or communicate, without being filled by an actor.

The interaction between actors can be divided up further, much in line with the interprocess-communication in PPM, but with additional possibilities for language action modeling. In addition to the type and contents of the items being sent, how many that they are sent to or received

by, and if they trigger the other actor into doing something, one can indicate the set of illocutionary points, propositional contents, and dominant claims of the communication. Each illocutionary act can in addition be related to a set of rules. We will discuss this last aspect in more detail after presenting the extended rule language in the next section.

Relationships between actors can be further annotated, by additional actors or roles on both sides of the relationship.

- *In the role of*: An actor can support, communicate, or be part of another actor in a given role. In the cases of communicate and support one can also annotate the receiving actor similarly.
- *On behalf of*: An actor can communicate, support, or be part of another actor on behalf of another actor. In the cases of communicate and support one can also annotate the receiving actor similarly. Using 'on-behalf-of' one can represent authorization.

Based on the actor class hierarchy in Figure 2.1, it is explicitly distinguish between the following types of actors.:

- Individual actor (I) e.g. you and me.
- Organizational actor (E) e.g. IDT, the institute.
- Applicative actor (A) e.g. The conference system C4.
- Supportive actor (S) e.g. Ingres DBMS.
- Hardware actor (H) e.g storlind (a file-server at IDT).

One can also indicate if an actor is external or internal relative to some organizational actor. The internal actor is typically the organization in which the COIS is being developed and maintained, but do not need to, i.e. one can easily produce different views of actor-models. The types of actors above can freely be mixed with the internal/external classification.

Roles can also have the types as indicated above, and can also be classified as external or internal, an internal role being typically a formal position in the organization.

The use of relationships are limited based on different actors:

Part-of relationship:

- An individual actor can only be part of organizational actors.
- All actors can be part of organizational actors.
- A hardware actor can only be part of another hardware actor or an organizational actor.
- A software actor can only be part of another software actor or an organizational actor.
- No actors can be part of an individual actor.

The support relationship has the following more specialized meanings.

- If one hardware-actor supports another hardware-actor it is used by it to achieve an overall goal that some social actor has set for it. General hardware compatibility is indicated by indicating *potential* support.
- If one hardware actor supports a software actor (supportive or applicative), the software actor is executed on it. General executional compatibility is indicated through a potential support.
- If one software actor supports another software actor, it is used by it to achieve an overall goal that some social actor has set for it. General software compatibility is indicated by potential support.
- If a computational actor supports a social actor, it means that it is used by the individual or organizational actor to support further goals.
- If a social actor supports a computational actor, then it sustains the possibility for the computational actor to support other social actors. Indirectly thus, these social actors support the other social actors.

- That a social actor support another social actor indicate that it help the other actor to achieve its goal in some way. This can be a general support, or be more specialized, e.g. by performing an activity, or by furnishing the other actor with necessary resources.

When actually supporting another actor directly, one always communicate with this actor, something which also can be done between all kinds of actors as indicated with support. It is also possible for actors to communicate without supporting each other.

Finally, it is possible to indicate the power-relationships between actors and roles, indicating the power structure in an organization. The relation can be general, or be specialized to e.g. the performance of a specific activity.

Some implications of the above structures, some of which can be utilized for consistency checking and driving questions are:

- The part-of relationship is transitive, if actor A1 is part of another actor A2 that is part of actor A3, then actor A1 is part of actor A3. The same applies between roles, if role R1 is part of role R2 that is part of role R3, the role R1 is part of role R3.
- If an actor A fills role R that has sub-role RS, then A also fills RS.
- If an agent is specified, the actor fills the specified role.
- If an actor support, communicate, or is part of another actor in a given role, it fills this role. The same applies on the actor that is supported or is communicated to on the receiving side.
- If an individual actor support, communicate, or is part of an actor on behalf of an organizational actor, it is part of this actor. The same applies to the actor that is supported or communicated to on the receiving side.
- The type of an actor indicates the main role of this actor, i.e. an organizational actor has the role “organization”.
- Every organizational actor OA implies the existence of a role “member of OA”.
- If an actor potentially support another, it means that they possibly communicate.
- If an actor is indicated to be internal to an organizational actor it implies that it is part-of this actor.
- If a role is indicated to be internal to an organizational actor, it is (partly) instituted by this actor.
- If an actor support an organizational actor, it potentially support each member of the organization.
- If a technical actor is internal to an organization, it can potentially support also the other members of the organizational actor.
- If an actor is supported by another actor in a given role, one can say that the organizational actor that this role is internal to is also supported by this actor.
- If an actor is supported by a role, then the actor is potentially supported by any actor that currently fill this role.

The above language features can be used to enhance modeling in PPP in several areas:

- Organizational modeling.
- Technical actor interaction modeling.
- Extensions of process modeling.
- Speech act modeling.

We will also briefly mention how the mechanisms above might be applied within user modeling and software process modeling in connection with project management.

Organizational modeling

Organizational models contains actors, roles, actor and role relationships, power relationships, support relationships, and agents.

An example of parts of an organizational model of IDT is given in Figure 10.6.

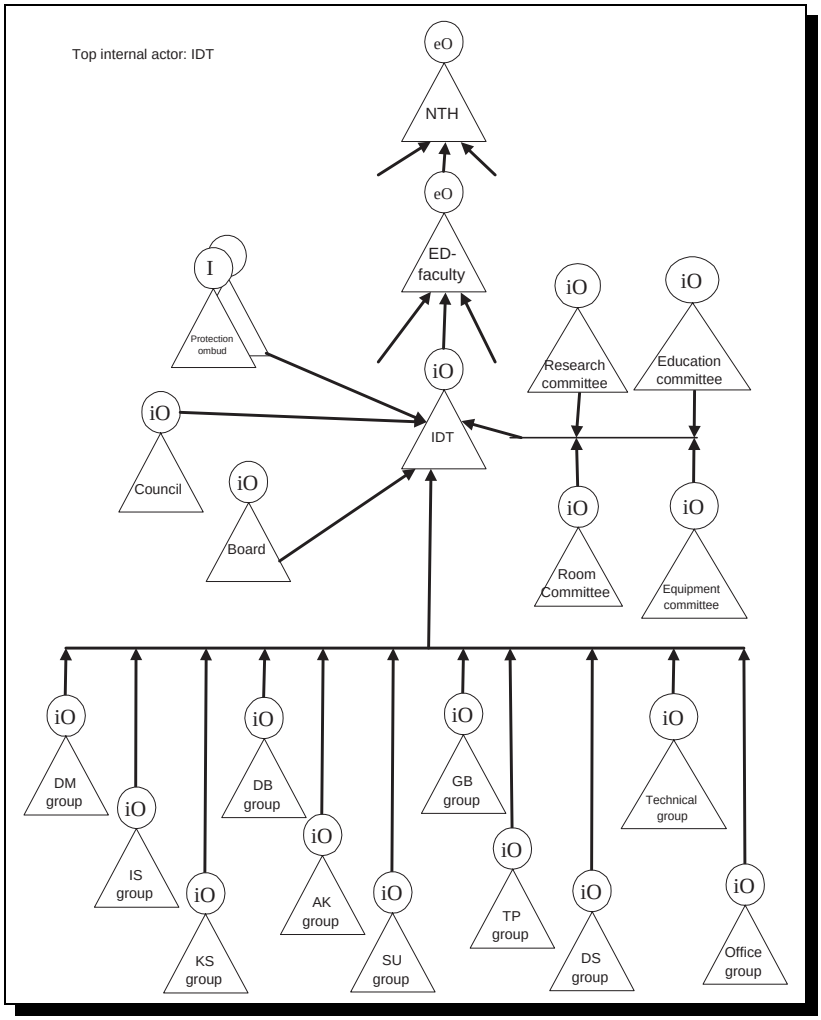

Figure 9.2: Overall organizational model of IDT

The institute has 11 main groups, shown in the bottom of the figure, anyone being employed at the institute being part of one. In addition there are several other groups and committees. Also one role is represented, being institutionalized by IDT.

Modeling of technical actor interaction

This can be used to indicate the interoperability of technical actors, to be able to give an overview over how the portfolio is intertwined on a high level, and how social actors are linked to this.

Different kinds of integration are possible to indicate on a high level in the following manner. The typology is based on Thomas [308]:

- Data integration: This is indicated with flows between actors. This can be specified further by indicating the type of the flow and the channel. One can also indicate more precisely

the interaction between actors, e.g. if it is synchronous or asynchronous, by either indicating the supportive actor supporting the flow, or use the store-symbol as used in PPM. Also multi-casting and broadcasting can be specified.
  - Multi-casting is indicated by splitting the flow at the output.
  - If the receiver is a role, we have broadcasting, meaning that the item might be received by all members of the role.
- User interface integration: Which user-interface, standards etc. that are used can be specified by indicating compatibility through the support relationship. In addition is it possible to indicate appropriate properties of the role "software actor".
- Control and process integration: Indicated by using the same control-interaction mechanisms as used in PPM, i.e. indicating if the flow is triggering or terminating. It is also possible to specify a more detailed input-output matrix for actors, similarly as for processes in PPM.

Extension of PPM

The PPM-language is extended in the following ways:

- External agents are either depicted as actors or roles.
- Actors and roles are further classified as illustrated above. This means that the use of the terms external and internal are different from how they are used in traditional DFD.
- It is possible to have flows going between actors and all the other constructs, not only from actors to and from processes.
- It is possible to specify for processes, stores, and flows which actors or roles that support the transformation, preservation, or transportation activities of these. One can choose if this should be shown graphically in the models. The default is that they are not shown. Neither is the possible 'in-role-of' and 'on-behalf-of' annotations to the relationships.
  When the automation border is drawn, this will be equivalent to saying that the processes, stores and flows within the border are to be supported by the new or updated applicative actor in the future. One can also distinguish this further, saying that whereas it is the application system that is responsible for some process, it might be a supportive actor (e.g. a database system, or the operating system) that is responsible for a store, i.e. supporting simple client/server modeling. When using an active databases, it is also possible to specify that the supportive actor is also to be responsible for the process, e.g. in the form of a database procedure. These kind of distinctions will typically only be used during detailed design.

Roles can be linked to entities in the ONER-diagram that can describe the potential state-variables of the actors filling these roles. This can be extended with indicating additional types for the single actor if necessary. The actors filling the role can either be all the instances of the entity, or the instances participating in a relationship that the entity is involved in.

Actors filling a set of roles will by default inherit the rules that applies to these roles, but this can be overridden. On the other hand, if a role is said to support e.g. a process this do only mean that actors of these roles *potentially* support this process. One can indicate capabilities as indicated in Chapter 7.4 both for roles and actors with the extension that one can say if the capability is potential or proven. Capabilities are inherited in the same way as rules. Also this inheritance can be overridden.

By the above support-mechanisms, an actor can have a view to a scenario indicating the part of a data-model it can access, and a set of rules that applies to it. Thus an actor might have access

to both it's own and also other actor's state and their potential and real capabilities and responsibilities.

Two examples of ways to indicate the support-relationships taken from the conference example are given in Figure 9.3 where the supporting roles are indicated in full in the diagrams, and Figure 10.10, where the supporting actor is indicated as being "behind" the process.

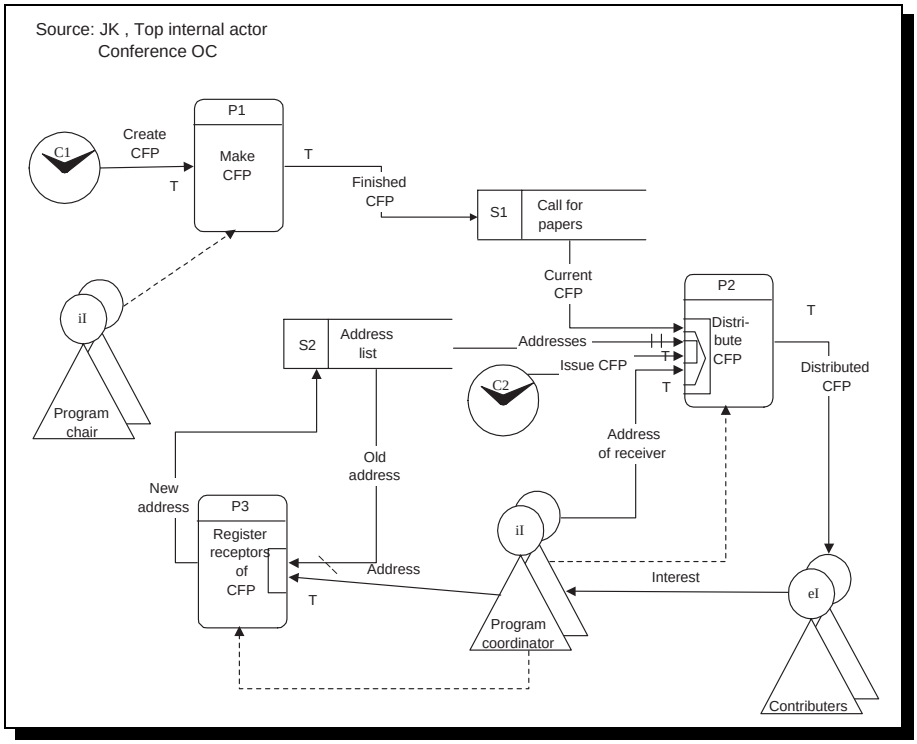

Figure 9.3: PPM describing creation and distribution of Call for Papers

Speech act modeling

More detailed modeling of the communication structure in the organization can be done using the suggested structure and language action feature.

The traditional five illocutionary points of assertion, direction, commission, expression, and declaration are supported, and are indicated with the abbreviations ASS, DIR, COM, EXPR, and DECL. The dominant claims can be claim to power, claim to truth, claim to sincerity, and claim to justice. As indicated in Figure 5.17, not all combinations of illocutionary points and validity claims are likely. One can also indicate a set of rules related to a speech act, covering among other things the condition of satisfaction. We will discuss this further after presenting the rule-language in Section 9.2.

On a high level, the general loop as indicated in Figure 5.16 can be depicted between actors as depicted in Figure 9.4. Dominant claims are not indicated. In addition to the flow-name that indicate the data or material that is sent, we indicate the propositional content to relate the flows in the same workflow-loop.

Figure 9.5 indicates the same situation, but being integrated with the extended PPM, such that the four main steps are indicated as processes, which then can be decomposed in the ordinary way

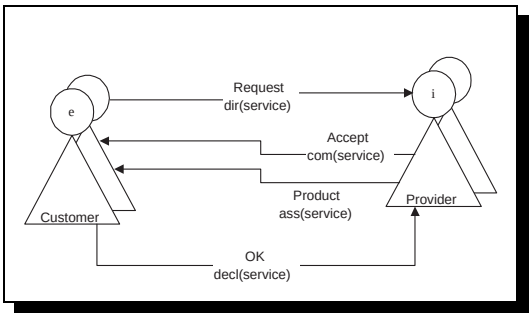

Figure 9.4: Simple use of illocutionary acts

or as further workflow-loops. If it is an organizational actor that supports a process, the actors that support the sub-processes of the process should be parts of this actor.

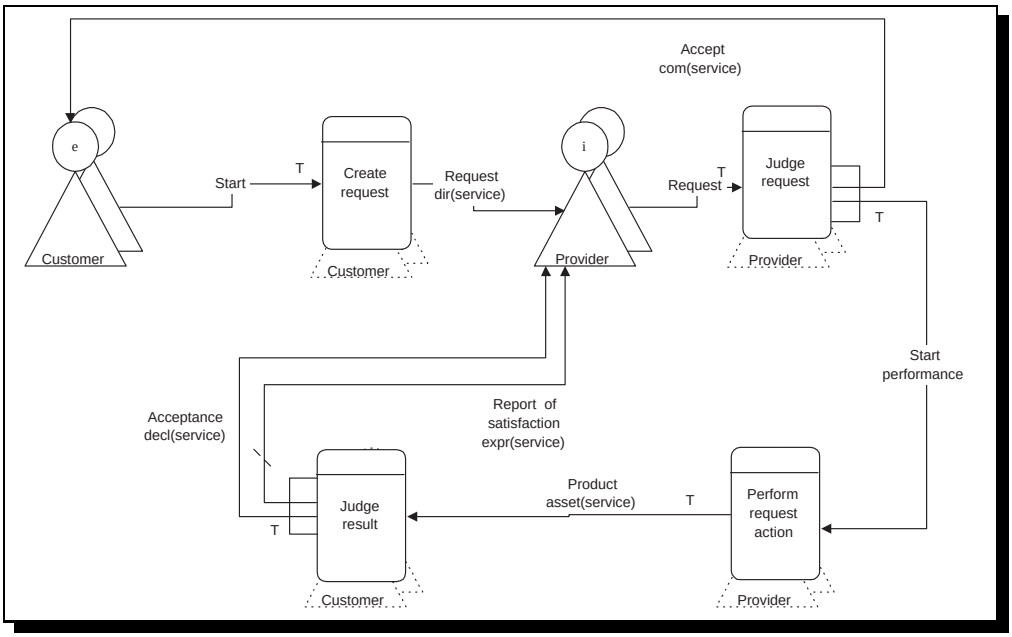

Figure 9.5: Illocutionary acts in PPM

Support for the general case as described in Figure 5.15 is depicted in Figure 9.6. When indicating dominant claim, one can also represent similar situations as in the extended model for ABC-diagrams as described in [314].

Note that the diagrams depicted here is specific diagrams, and many other communication-structures can be modeled. We can also use other modeling constructs such as stores and timers if necessary. Thus, it is possible to model a much greater variety of communication structures than in action workflow. On the other hand, one lose much of the specific support for modeling according to this structure.

The modeling of speech acts in SAMPO as depicted in Figure 7.12 and 7.13 can likewise be supported as depicted in Figures 9.7 and 9.8 respectively. Our models depict a slightly different situation to be closer to the conference case.

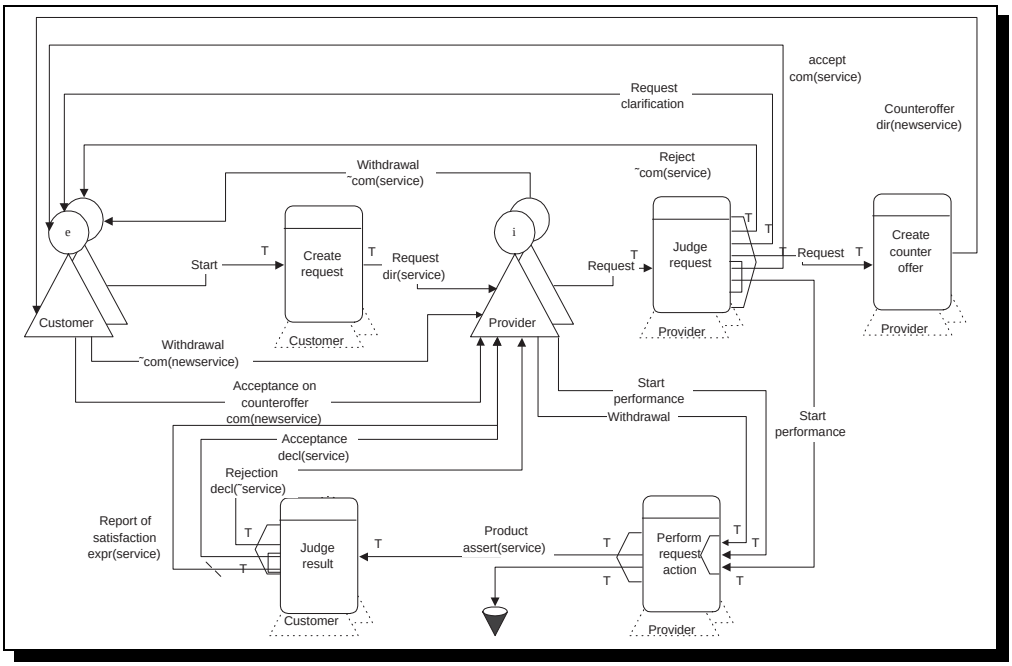

Figure 9.6: Conversation for action in PPM

User model

As has already been described in Chapter 7.4 user models are applied in different areas in connection to explanation generation and can be modeled as an extension of the individual actors or be connected to the identified roles, thus supporting both individual and canonical user-models. In connection with explanation generation, the following relationships are suggested:

- The user characterization submodels, which characterizes the user in terms of which elements of the source model are accessible to him are modeled by state-variables linked to the actors being part of the audience.
- The user knowledge sub-model, which specifies the parts of the source model that are known to the user. This will be modeled by the knowledge the individual actor have about the model and language model respectively, functioning as overlay models to these models.

Since user modeling is a whole discipline in itself and depends on extensive analysis of user behavior and background, we regard this as being beyond the scope of this thesis and leave the subject here.

Project management

This can be supported by specifying roles, responsibilities, explicit and potential capabilities, and access to data for the participants in the project. This will partly overlap with the user-modeling as suggested above. The previous version of PPP has been evaluated for its usefulness in software process-modeling in [57] and some, but not all of the identified weaknesses can be addressed by the possibility of comprehensive modeling of actors. A way of supporting cooperative development of conceptual models is presented in [4].

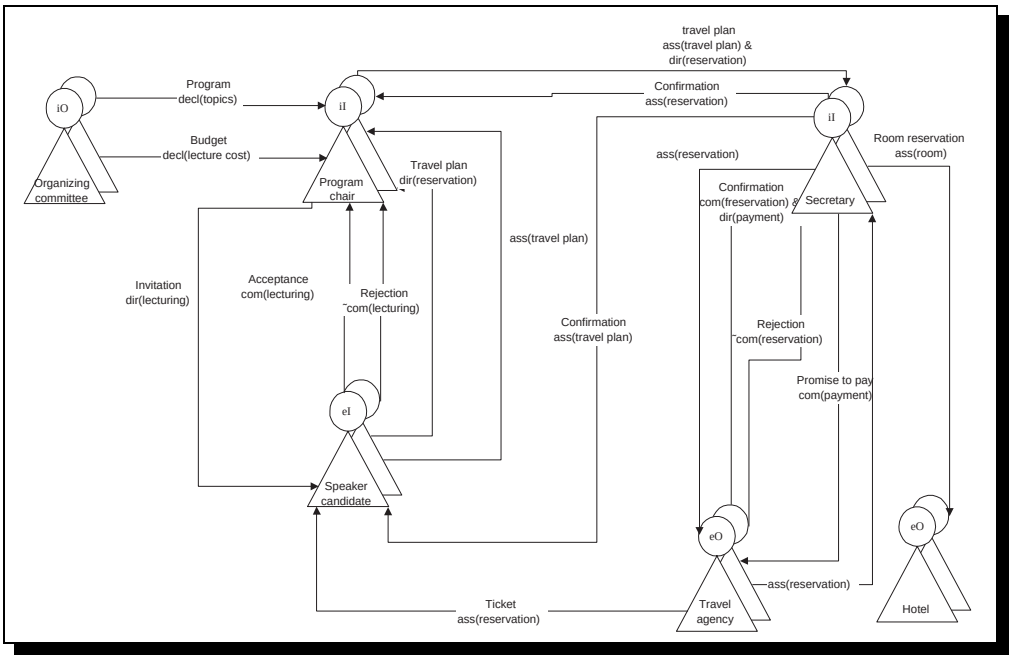

Figure 9.7: Invitation of speakers in PPP

9.2 DRL: Deontic Rule Language

The Tempora ERL as described in Chapter 7.2 extended with deontic operators and roles will be included in our set of modeling languages. Potential positive aspects of rules for supporting a social construction process and maintenance were touched upon briefly in Chapter 5.2.4. Especially by including deontic operators in and between rules, and thus capturing the deontic statements that have entered into the argument leading to the current solution, including rules in the conceptual framework is looked upon as important in the a long term process to avoid premature closure. High level rules might also be useful on the portfolio level to link applications that are used to fulfill parts of higher level organizational goals. These kind of links is also regarded to be useful from a reuse point of view, giving more information about the context in which a given artifact is currently used.

How to integrate the ERL in PPP has earlier been discussed by Seltveit [282]. The integration is made easier by the fact that the process modeling language in Tempora is an adaption of the PPM language. The link between these languages was originally described by us in [168, 179].

According to Seltveit, there are at least two possibilities for including dynamic rules in the PPP languages:

- Substitute rules for PLD for describing process logic.
- Allowing specifications expressed using rules and PLD to co-exist.

Although Tempora has shown to be quite successful in dealing with rules, experience also show that rules may be difficult to formulate and especially dynamic rules tend to become very complicated and difficult to comprehend [282]. Therefore one should take advantage of the user-friendliness of the PLD language for specifying procedural aspects and the advantage of ERL for specifying declarative aspects. Thus the second option is chosen. This means that a process in a PPM can be associated with: A set of decomposed processes, a PLD description, or a set of rules. In addition an input/output matrix can be specified. The DRL must use ONER instead of

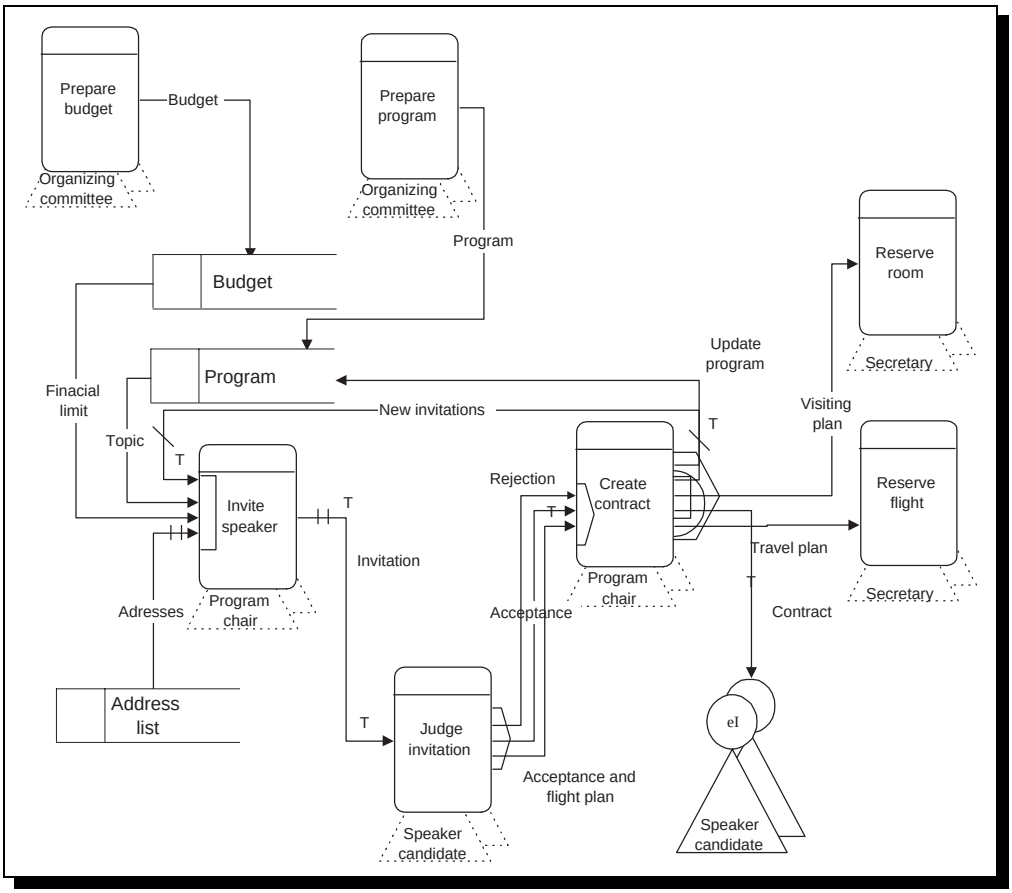

Figure 9.8: Detailed invitation of speaker in PPM

ERT. Since ONER do not have constructs for specifying that the history of an entity should be retained, parts of the temporal expressiveness of the rule language used for querying a historical database will not be applicable without further extensions of ONER.

In addition to incorporating ERL as is, the rule-language is extended by introduction deontic operators and roles/actors as outlined in [180].

As most rule-languages, ERL can only express that some state or action is required or prohibited, and it is impossible to distinguish between rules of necessity and deontic rules. Several newer approaches to rule-modeling addresses this distinction [46, 329] and the use of deontic operators in rule languages have increased lately [157, 223]. Deontic operators have been used in connection to security, authorization, database integrity, requirement specification and conceptual modeling, text analysis, and conflict resolution to mention some.

Our proposal of deontic operators goes in a similar direction as the suggestions for extending deontic logic by Jones and Pörn [154, 155, 156]: In addition to distinguish between rules of necessity and deontic rules there is a need to distinguish between the ideal and the sub-ideal. We thus propose to operate with three different kinds of rules:

- Rules of an analytical or empirical nature, which will be written without deontic operators.
- Rules about actions or states which are *obligatory*, *permitted*, or *forbidden.*
- Rules about actions or states which are *recommended* or *discouraged.*

As an example, consider the speeding legislation of some road. According to the generally agreed definition of speed, which is positive either way, it is impossible to go at negative speed. Accord-

ing to the theory of relativity, it is impossible to go faster than the speed of light. These rules constitute the main analytical and empirical rules of the situation. Further, it may be forbidden to go faster than, say, 90 km/h. For a certain road it might still be legal to go at 90, but recommended not to go faster than 70, e.g., because of sharp curves. Finally, there may be an exception, e.g., a rule permitting police, doctors, and priests to go faster than 90 in situations of emergency. This shows a possible use of the permitted operator, and at the same time the need to connect rules to roles: actions that are permitted to actors of some roles (e.g. a policeman pursuing a criminal) may be forbidden to others (e.g. a tourist, or the same policeman when he is off duty).

9.2.1 Intra-rule Deontic Operators

The extended rule-format is:

> when *trigger* if *condition* then it is *deontic* for *role/actor consequence*
> else it is *deontic* for *role/actor consequence*

- *trigger* is optional. It refers to a state change, i.e. the rule will only be enabled in cases where the trigger part becomes true, after having been previously false. The trigger is expressed in a limited form of first order temporal logic.
- *condition* is an optional condition not referring to a state change in first order temporal logic.
- *deontic* is one of the deontic operators *obligatory* (**O**), *recommended* (**R**), *permitted* (**P**), *discouraged* (**D**), and *forbidden*(**F**). A rule of necessity has no deontic operator.
- *role/actor* is an optional specification of the role of those being subject to the rule, i.e. the characteristics of the one being responsible for achieving or avoiding the consequence of the rule. The default role is *everyone*. Roles and actors can be further described using the actor modeling language.
- *consequence* is an action or state which should hold given the trigger and condition. The consequence is expressed in a limited form of the rule language. (For instance is not disjunctions allowed in a consequence. Negations are not used on an action). The 'else' clause indicates the *consequence* when the condition is not true, given the same trigger.

As indicated in the previous section, a rule applying to a role also applies to the actors that currently fill the role. On the other hand, it is possible to overrule this specifically by indicating rules applying for the actor. Generally, rule applying to an actor or a super-role of a role will have precedence of a rule applying to the role. This is similar to the principle of specificity [272], i.e. that normative statements about a smaller class of objects override those about a bigger class. To be more general, one should also include a mechanism for role-overruling as suggested in [54].

As discussed in Chapter 7.4, complex delay and clock intervals can be expressed by rules. A general outline is given below.

- Clocks: One rule for each on-flow:

```
When temporal expression
[if onflow([p1...])]
[and not offlow([p1...]) [since onflow([p1...])]]
then outflow1([p1...]) [and outflow2([p1...])]
```

- Delays: One rule for each on-flow:

```
When temporal expression since onflow(p1,p2...)
[if not offlow(p1,p3....) since onflow(p1,p2...])]
then outflow1(p1,p2....]) [and outflow2(p1...)]
```

It should also be possible to indicate similar rules for actors and roles for simulation purposes. This is not rules applicable to the actor or role. One should then indicate also here type of flow (on,off). We will not pursue this further here.

When using rules, one will be able to express the rules both in an informal manner and in DRL if necessary. The informal expression should follow the general rule-format. Note that since it is only the *consequence* that is mandatory, the rule format also cover the expression of high-level goals. In addition should it be possible to represent when the rule was defined, which actor has defined it and formalized it, which actor was the source of the rule, and which actors agree to the rule. It should also be possible to represent the instituter of the rule. Rules also have an id and a short name as in Tempora. A rule has a current status, similarly to all model elements. A rule can also be classified as functional or non-functional. We will discuss the use of this data further in Chapter 10.

9.2.2 Inter-rule Deontic Operators

We also suggest the use of deontic operators *between* rules. In the above situation, there might be higher level rules that traffic should be safe and that traffic should be effective. The rule about safety might recommend the lower level rules limiting the speed. At the same time, the rule about effectiveness may recommend the exception for emergency cases, as well as prohibit too restrictive speed limits.

In addition to the inter-rule operators already existing in Tempora, we propose *permits*, *discourages*, and *forbids* as additional operators in the goal-hierarchy. Whereas one has made positive experience in Tempora with the goal-hierarchies [281, 282, 324], also some problems have been reported [282]:

- Each developer seemed to have different opinions about the meaning of the relationships. This inspire us to come up with a firmer although not formal definition of the links. The reason that we do not suggest a formal semantics of the relationships is that they will be used to link informal rules and goals. In addition to the links and the operators themselves, similarly to rules, all links have a source, a capturer, a set of people agreeing to the link, and a formalizer. Finally, it is possible to annotate links with arguments.
- Maintenance of non-automatic links between rules was difficult. In Tempora, there was in reality no tool-support for doing this (and neither any tool-support for showing rule-hierarchies), thus to be able to use large rule-hierarchies, tool support seem to be of necessity.

The names of the relationships are changed from those reported in connection with Tempora to suit a deontic mind-set.

- *Necessitates* and *exclude*: Used to link rules where definitions of terms or the introduction of a necessary situation can be found in another rule, and where the consequences of two rules are inconsistent respectively.
  The *necessitates* relationship is similar to the *refers-to* relationship discussed in [281]. Also the relationship *caused-by*, *uses*, and *has-info-derived-by* as suggested in [282] can be said to be different kinds of necessitations. More precisely, we can say that if **R1** necessitates **R2**, this is equivalent with the rule:

  > If one wants to fulfill **R1** it is necessary to fulfill **R2**

  If a rule is a rule of necessity, an obligation, a recommendation, or a permission and an authorized actor wants to fulfill the rule, fulfillment of the rule means that the consequence

of the rule is made true if the condition of the rule is made true. If a rule is a discouragement or a prohibition, fulfillment of the rule means that the consequence of the rule is not made true if the condition of the rule is made true. As illustrated in the causes-relationship below, the condition is not necessarily sufficient.
The different variants of necessitations are all automatically maintained and are described below:

- *Causes:* Denotes a causal relationship and corresponds to that a term appears in the consequence of an event-action rule **R1** and in the trigger of another rule **R2**. Thus for the second rule to ever trigger, the first rule must exist. The inverse relation is *caused-by*. These relations are implicitly specified when linking rules to the process model.
- *Uses:* Links a rule and a predicate rule used by that rule, the inverse of uses is *used-by*. Also this relation is implicit.
- *Has-info-derived-by*: Links a rule and a derivation rule. The first rule contains expressions that are derived by the derivation rule.

For exclusion, we have for the rule "**R1** exclude **R2**":

> If one wants to fulfill **R1** it is excluded to fulfill **R2**

This is a symmetric relationship.

- *Obligates*, *recommends*, *permits*, *discourages*, and *forbids*: These relationships are used to create rule-hierarchies that can be used for rule externalization, rule analysis, internalization, argumentation, inconsistency-modeling, representation of variety and rule maintenance.
  - **R1** *obligates* **R2**: This is defined with that if one are supposed to achieve the higher level goal **R1**, those agreeing to the link regard that one must also perform the lower level goal **R2**. Stated as a rule, this reads:

    > If one is to fulfill **R1** it is obligatory to fulfill **R2**

  - **R1** *recommends* **R2**: This is also used to tell why we have a lower level rule, but is weaker than *obligates.*
    Stated as a rule, this reads:

    > If one is to fulfill **R1** it is recommended to fulfill **R2**

  - **R1** *permits* **R2**: One rule, being institutionalized by an actor either internal or external to the organization permits one to have another rule. Thus, if **R1** is removed or changed, this might invalidate **R2** even if **R1** do not give any direct motivation for having **R2**.
    Stated as a rule, this reads:

    > The fulfillment of **R2** is permitted by the fulfillment of **R1**

  - **R1** *discourages* **R2**: Can be used in cases where different rules are regarded to work against each other, something which is important to capture to be able to discuss under what circumstances one or the other policy should be given precedence.
    Stated as a rule, this reads:

    > If one is to fulfill **R1** it is discouraged to fulfill **R2**

  - **R1** *forbids* **R2** : This is an even stronger relationship and should be resolved if both the rules are to be automated. One possibility when both rules are regarded to be necessary is to transfer the relationship to the exception-hierarchy described in Chapter 7.2, by specifying the direction and kind of exception to apply.

Stated as a rule, this reads:

If one is to fulfill **R1** it is forbidden to fulfill **R2**

Both *discourages* and *forbids*-relationships can be symmetric.

- *Overrules* and *suspends*: These are used in the same way described in Chapter 7.2.

Early in modeling, the effect two rules have on each other will not always be apparent, only that they seem related in some manner. In this case a more informal *related-to* relationship as suggested in [282] can be used. Only the exception-hierarchy is utilized directly by the CIS during run-time.

In addition to the relationships, the following mechanisms are used to create a directed acyclic graphs with and/or-nodes of rules. The description below refers to Figure 9.9.

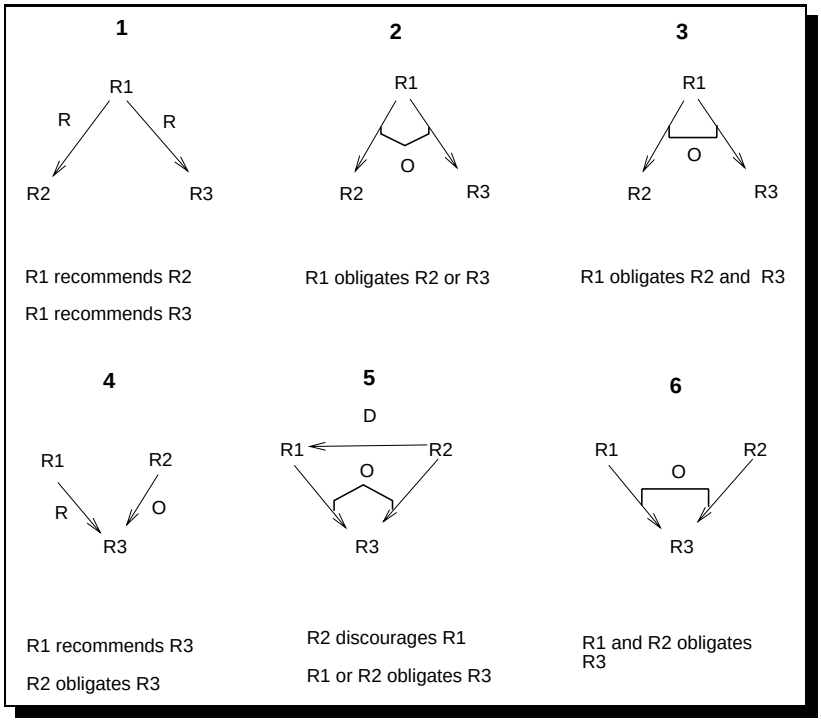

Figure 9.9: Mechanism in rule-hierarchies

- 1. Rule **R1** recommends rule **R2** and **R3** independently.
- 2. Illustrates an or-situation. Based on rule **R1**, there are two perceived ways of partly fulfilling this in a lower level rule. These two rules, **R2** and **R3** might be inconsistent, even if this is not indicated in the diagram.
  Stated as a rule, this reads:

  If one is to fulfill **R1** it is obligatory to fulfill only_one_of (**R2**, **R3**)

  The rules **R2** and **R3** might be two rules suggested by the same actor, or it might be two different actors view about how rule **R1** should be partly fulfilled. If the different rules **R2** and **R3** is agreed upon by different persons, and these are regarded to discourage, forbid, or exclude one another, one has identified an *issue*, with **R2** and **R3** being initial *positions* to the issue. Other deontic links to the positions will be *arguments* thus one can use IBIS[1]-

[1]Issues Based Information Systems.

like techniques as originally suggested by Rittel [268] as part of creating the goal-hierarchy. How this can be done in practice is illustrated in Chapter 10.

- 3. This is similar to the first situation, but indicates that both **R2** and **R3** should be fulfilled simultaneously if one want to fulfill **R1**, thus **R2** and **R3** are subgoals of **R1**. Stated as a rule, this read:

  If one is to fulfill **R1** it is obligatory to fulfill **R2** and **R3**

- 4. Indicates that rule **R3** is both recommended by rule **R1** and obligated by rule **R2** independently.
- 5. Indicates that rule **R3** is obligated by either rule **R1** or rule **R2**, but not both, since in this case, having rule **R2** in place discourages having rule **R1**. A discouragement where **R1** and **R2** are supported by different actors is another situation where an issue can be identified. Stated as a rule, this reads:

  If one is to fulfill only_one_of(**R1**,**R2**) it is obligatory to fulfill **R3**

- 6. Similar to the fourth situation, but indicates that it is the simultaneous fulfilling of **R1** and **R2** that obligates **R3**. The reason for modeling this kind of situation is that one might choose **R1** not to be valid of some reason. With this situation, **R3** would not be regarded to be obligated.
  Stated as a rule, this reads:

  If one is to fulfill **R1** and **R2** it is obligatory to fulfill **R3**

One can further combine these mechanism if necessary. Both functional and non-functional goals and rules can be part of the goal-hierarchy. When to introduce functional and non-functional goal should be taken care of in the methodology assuring that the approach will be problem-oriented and not product-oriented in the early stages to the extent possible.

The syntax of the extended rule-language including rules for inter-rule operators and timers is given in Appendix C. The syntax is based upon the syntax of Tempora ERL.

An example on the usage of the intra and inter-rule deontic operators is illustrated in Figure 9.10. The example is taken from the Sweden Post-case study.

The rules in an informal form are as follows:

- **R1**: "It is *obligatory* for the Post to achieve a satisfactory profit".
- **R2**: "It is *obligatory* for the Post to get paid quickly".
- **R3**: "It is *obligatory* for the Post to get paid for services".
- **R4**: "It is *obligatory* for the Post to offer good service to their customers".
- **R5**: "It is *obligatory* that reminders are sent out when invoices are not paid within the due date".
  Note that when specializing this rule even further, we get the action-rule shown in Figure 7.7 that is linked to process P9 in addition to other rules not shown, thus showing an example of the relationship between high-level goals and low-level implementation.
- **R6**: "It is *forbidden* for the Post to take good customers to court".
- **R7**: "If a bill is not paid when two weeks have passed after a company has issued the second reminder on the invoice, it is *permitted* for the company to take the claim to court".
  In contrast to the other rules, the institutionalizer of this rule is not the Post, but an external actor, in this case the Government.
- **R8**: "If a customer has not paid his invoice two weeks after that the second reminder on this invoice has been issued, it is *recommended* for the Post to take the case to court".

The relationships in this part of the example are:

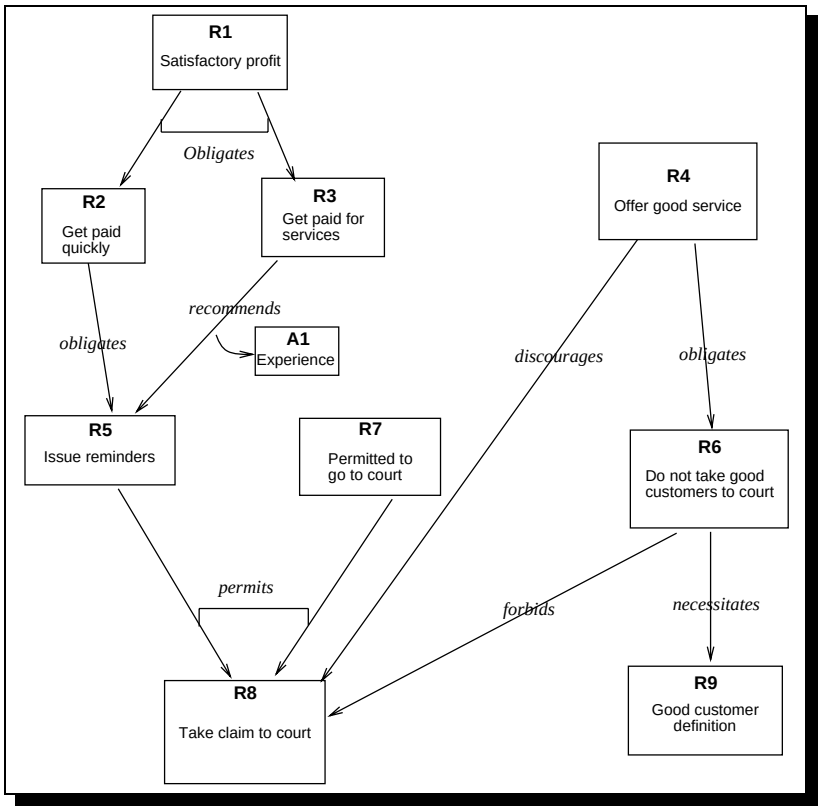

Figure 9.10: Example of a rule hierarchy

- **R1** *obligates* **R2** and **R3**.
- **R2** *obligates* **R5**.
- **R3** *recommends* **R5**. This is supported further by argument **A1** "Based on experience, not all customers pay their dues if they are not reminded".
- **R4** *obligates* **R6**.
- **R6** *necessitates* **R9**. This is one example of a necessitation relationship, where the use of the term "good customer" necessitates a rule that defines what a good customer is.
- **R5** and **R7** *permits* **R8**. Both rules need to be present here to be able to have a rule about taking claims to court.
- **R4** *discourages* **R8**.
- **R6** *forbids* **R8**.

9.2.3 Speech Acts and Deontic Rules

As indicated in the last section, it is possible to link deontic rules to speech acts. As discussed in [75], there appears to be a connection between deontic rules and illocutionary logic. Based on the example depicted in Figure A.27, we will describe more fully the communication between a conference organizer, and an IS-professional in connection with the paper- process to exemplify the modeling of this and also the modeling of validity claim which was not included in the examples in the last section. There is normally no power-relations between these parties, thus all claims will be either to truth, justice, or sincerity.

The presentation is based on the sending of items. The illocutionary acts is then presented as a

triplet <illocutionary point, propositional content, dominant claim> and implied rules are listed after the illocutionary point if any, followed by a comment.

- Organizer distribute CFP to IS-professional.
  - < dir, paper, truth >

    ```
    If before CFP-deadline
    it is permitted for IS-professionals to issue a paper to the conference

    If issuing a paper to the conference
    it is recommended for the issuer to write
    within the areas of interest indicated in the CFP.
    ```

  - < ass, conference, truth >
- IS-professional *Person* sends a letter of intent to the conference organizer.
  - <com, paper, sincerity >

    ```
    when letterofintent(Person,Conference)
    if before CFP-deadline
    and Paper written by Person  has not been issued to Conference
    it is recommended for Person to issue a Paper to Conference within the
    CFP-deadline
    ```

    Note than in most cases where an offer is accepted, this will mean that an obligation is established. A letter of intent is requested in some conferences, but not in the one in the case-study, thus this part is not covered by the process model. This is one example of a workflow-loop which do not have all the four major steps.
- IS-professional issues a paper to the conference
  - < ass, paper, justice >
    By doing this, the person is no longer recommended to issue a paper to the conference if he had sent the letter of intent. The retraction of the recommendation is already covered by the above rule.
  - < com, presentation, justice >

    ```
    when issuepaper(Authors,Paper,Conference)
    if Paper accepted
    it is obligatory for at least one of the Authors to
    attend the Conference and present the Paper there.

    when issuepaper(Paper,Conference)
    and not withdraw(Paper) after issuepaper(Paper,Conference)
    it is obligatory for the Conference organizers of Conference
    to ensure the fair review of the  Paper
    ```

- The conference organizers issues a confirmation on that the paper is received:
  - <ass, confirmation, truth >
- The IS-professional withdraws his paper from the conference:
  - < decl , not paper, justice >

    ```
    When withdrawal(Paper)
    if Paper distributed to Reviewer
    it is obligatory to notice the reviewer of the changed situation.
    and update overview of  reviews to be expected.
    ```

- The conference organizers issues a rejection letter, including the review-forms
  - <decl, not paper, justice>
  - <expr, paper, sincerity> (the actual reviews)
- The information about conditional acceptance of the paper, including review-forms and copyright-form and style-guidelines are sent from the conference organizer to the IS-professional.
  - < decl, paper, justice >
  - < expr, paper, sincerity > (the actual reviews)
  - < dir, CRC, justice >

    ```
    If the paper is to be printed in the proceedings
    it is obligatory for the Authors to make and return a
    CRC following the style-guide
    and taking the review-comments into account within the CRC-deadline.
    ```

  - < dir, copyright, justice >

    ```
    if the paper is to be printed in the proceedings
    it is obligatory for the authors to pass over copyright to the publishers.
    ```

  - < dir, presentation, justice >

    ```
    it is obligatory for at least one of the authors to attend the
    conference and present the paper.
    ```

    Note that this is already promised by the authors above, thus here the commissive act in fact preceded the directive act

  Here, one possible situation is that the authors feel that some of the comments by the reviewers are impossible to adhere to and thus might come with a counter-offer based on truth, where they say what they will change.
  - < dir, CRC, truth >

    If this is accepted it is indicated through an acceptance letter < com, CRC, justice > from the conference organizers. The specific rule produced involving the updated condition of satisfaction will in this case overrule the general rule on the CRC's directly since it applies on an actor level.

  For usual (non-conditional) acceptance, the pattern is equal to above, with the difference that in this case would changing the paper according to the reviewers comment only be a recommendation, whereas the change according to the style guide is still an obligation.

9.3 Integration with PPP

We have so far only sketched the need for extensions of the other parts of PPP to be able to integrate our conceptual extensions with the already specified techniques for ensuring modeling quality.

We will here discuss the following:

- Extension of the meta-model to specify the repository support.
- Extension of prototyping.
- Extension of consistency checking.

This will include the main discussion on the executional semantics of the part of the language-extensions that are to be used during execution. In addition will the technique of using driving questions be outlined.

In Appendix D the additions to the following techniques are described based on the same executional semantics:

- Extension of the general execution facilities. This includes support of rules including the deontic operators and roles/actors and timers.
- Extensions of the explanation generation facilities. In addition to the extensions for execution, explanation generation is also extended to be able to utilize parts of the rule-hierarchy.
- Extending the filtering mechanisms to include the rule-hierarchies and actor models.

Extensions of the integrated tool-support will also be discussed briefly.

9.3.1 Extension of the Meta-Model

The existing repository structure of PPP as developed by Yang [340] is shown in Figure B.1. To accommodate rules, actors, and roles as described above, this structure must be extended. We will below indicate several scenarios of this updated model. We have also changed some of the terms used in the meta-model to be consistent with our terminology, thus the actual changes are not as large as one might get the impression of at first glance.

Figure 9.11 shows first a generalized extension to the meta-meta model of the tool, where it is indicated that for all model-elements (not only for rules as indicated above) it is possible to specify who defined the element, who have performed the formalization of it, who agrees to it, who is the source of it, and who is interested in it. It is also possible to indicate the actor having knowledge of a model element, and in addition which elements of the meta-model each actor is believed to have knowledge about. The two last relationships can be used to support the user-modeling facilities as part of user tailored explanation generation.

The status of a model-element indicate if it is temporarily refused, undecided, or accepted as part of the official model.

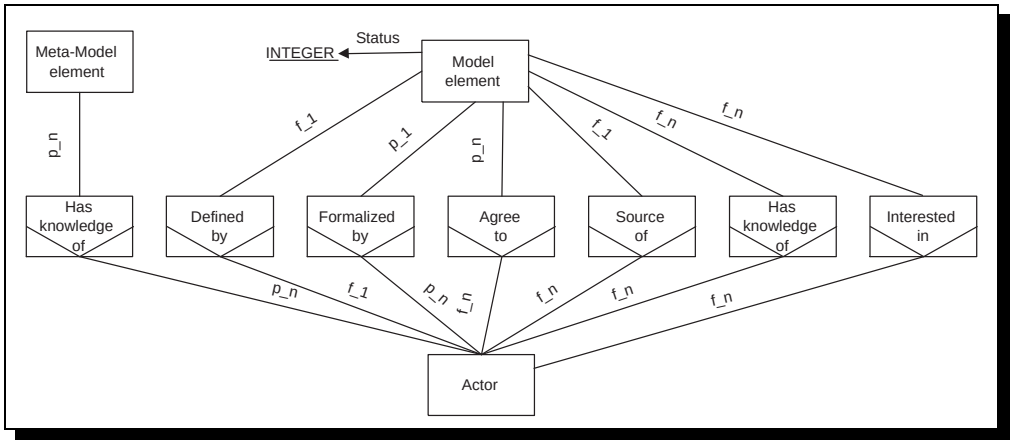

Figure 9.11: Extensions of the meta-meta-model in PPP

Figure 9.12 shows a ONER-scenario which include the main parts of the extensions as they relate to process and rule-modeling. Repeated entities are shaded. We have not indicated which entities should be versioned as in the figure of Appendix B and only the main attributes are included.

The main new entities that are changed or added are:

- DRL-rule: A DRL-rule has an ID, a type (i.e. constraint, derivation rule, action rule , predicate, non-functional or undefined), and a short-name. The rule can be stated as a formal DRL-rule or as a natural language (NL) expression or both. The DRL-rule can be further divided into the trigger, condition (which can refer to a scenario of a ONER-model) and a (set of) consequences. This is not indicated in the figure. In addition can it have one of the five deontic operators, and can apply to a role or an actor. It is also possible to capture which actor that has institutionalized the rule. This is not necessarily the same actor as the one that is the source of the rule in the meta-meta-model.

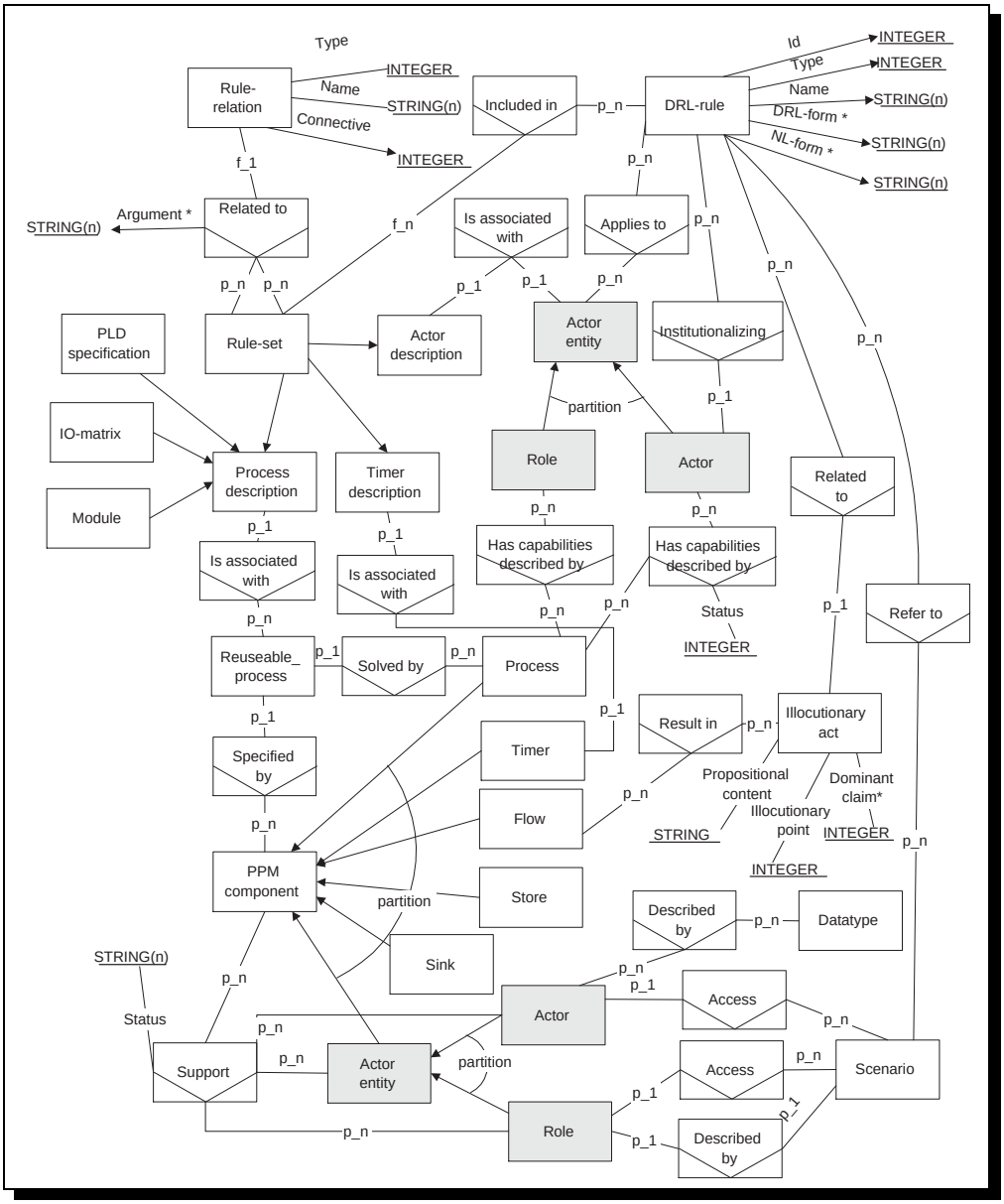

Figure 9.12: Incorporating the use of rules and actors in the PPP meta-model

- Rule-set: The clustering of rules into rule-sets is necessary for general rule-relations to be specified. The *rule-relation* has a type, indicating the type of relation (necessitate, obligates etc), a name, and a connective (and/or/none). The related-to relationship links a relation with two rule-sets, and can have an optional argument.
- The capabilities of actors and roles are described by processes. For the capabilities of an actor it is also possible to indicate the status, i.e. if they are proven or only potential. An actor or role can be said to support a set of PPM-components. An actor can access data described in a scenario of a data model, and can be described by a set of data-types in addition to the ones indicated through the link between the roles they fill, and the how these roles are described statically in the ONER-model. If an actor support a store, it is implicitly given that the actor can access the data described in the scenario for the store. An actor-entity (actor or role) can finally be associated with a set of rules, used in model simulations.
- Some changes to the existing phenomena classes of PPM are also performed: Processes and timers are potentially associated with rule-sets for their description. Flows are optionally connected to a set of illocutionary acts, for which one can indicate the propositional content, the illocutionary point, and the dominant claim. An illocutionary act can further be related to one or more rules.

In Figure 9.13 the actor-specific relationships are indicated. These relates to the description above, which is briefly re-described here following the figure from the top and downwards:

- An actor and a role can be related in an agent relationship. An agent can be institutionalized by an actor.
- An actor can fill several roles, and a role can be filled by several actors
- A role can be institutionalized by several actors, and an actor can institutionalize several roles.
- A role can be regarded as external to an (organizational) actor.
- A role can be regarded as internal to an (organizational) actor.
- An actor can be regarded as internal to another actor in a given role.
- An actor can be regarded as external to another actor in a given role.
- An actor can be part of another actor, and can be it on behalf of another actor, or in a certain role.
- An actor can in addition be linked to another actor through a support relationship, or a communication relationship, which again can be described by a flow. The link has an actor in the start and the end, and possible a start and end ‘on behalf of’-actor and an start and end role.
- An actor can be in a power-relation to another actor, possibly in given roles, and possibly in connection with given activities.
- Finally, it can be indicated that a role can be part of another role, and can overrule another role, possibly specifically for one or more actors.

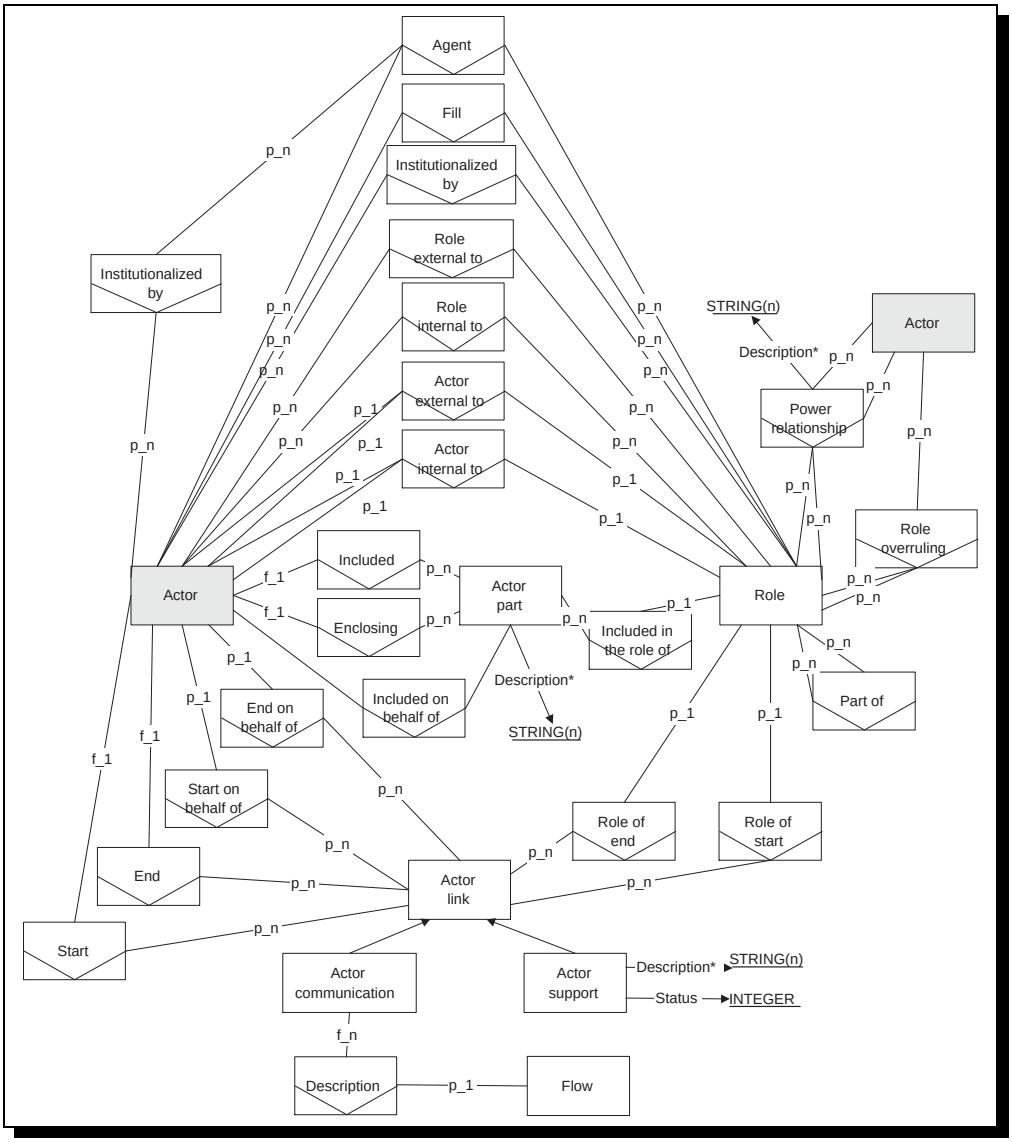

Figure 9.13: Meta-model based on the actor-modeling language

We have also included an update of the meta-model of the main entities of ONER, to indicate how the DRL-rules are integrated, and especially the extended way of describing methods. This is illustrated in Figure 9.14.

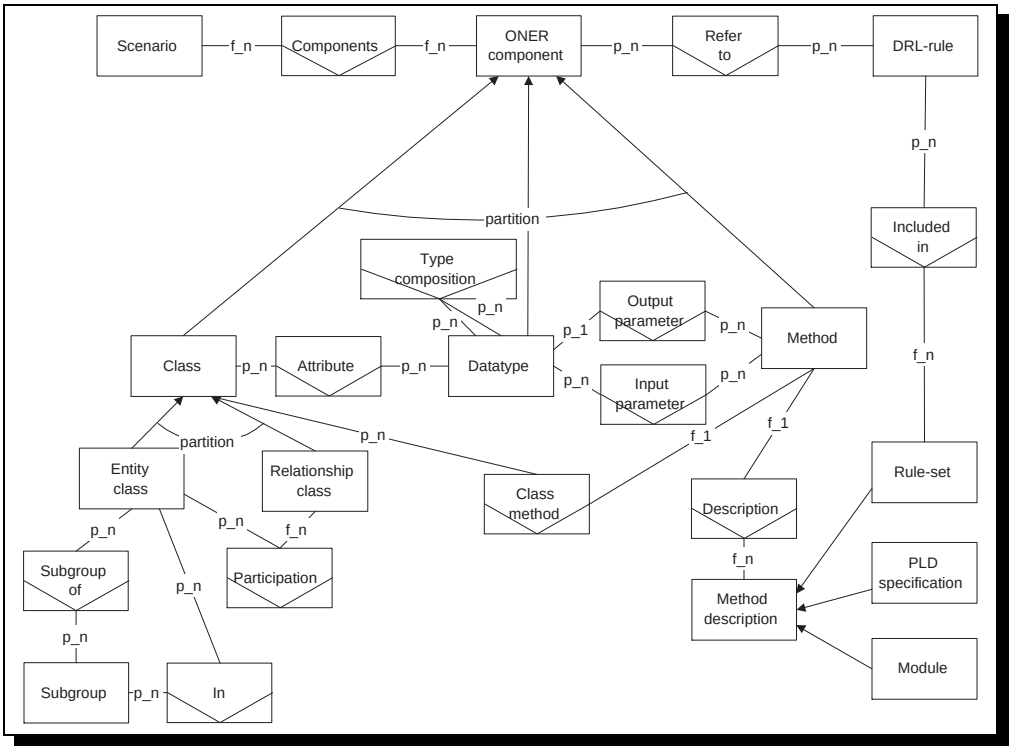

Figure 9.14: Updated meta-model for ONER

Finally, it is indicated how the overall relationship between the rule-hierarchies and the argumentation-structure as described in Section 9.2 can be supported in Figure 9.15.

9.3.2 Extension of Prototype Generation

As described in Chapter 7.4, there are several existing prototype generation systems in connection with PPP. The most comprehensive of these are the one that generates code in a combination of C and TEQUEL. This is also the easiest to extent taking our extensions into account. It already uses the run time environment of Tempora, and the design of this is prepared to create code when some of the bottom-level processes are described as PLD's and some are described using rules, applying the translation of ERL-rules to TEQUEL developed in the Tempora project. The addition of deontic operators and roles and actors in the rules can be supported by changing the rule execution algorithm of the Tempora run-time system as described below.

Omitting the 'else' clause, a DRL rule can be written:

when τ if ϕ it is ∇ for ρ $\psi \equiv \nabla_\rho$ $(\psi$ / $\neg \bullet \tau \wedge \tau \wedge \phi) \equiv \nabla_\rho(\psi$ /$\mathcal{E})$ [2]

[2] $\mathcal{E}$ is used as a shorthand for $(\neg \bullet \tau \wedge \tau \wedge \phi)$ below. The semantics of the trigger do not cater for the situation where the same trigger applies in two consecutive states, but this situation is unlikely to happen, thus we have used the same formalization of this as in Tempora.

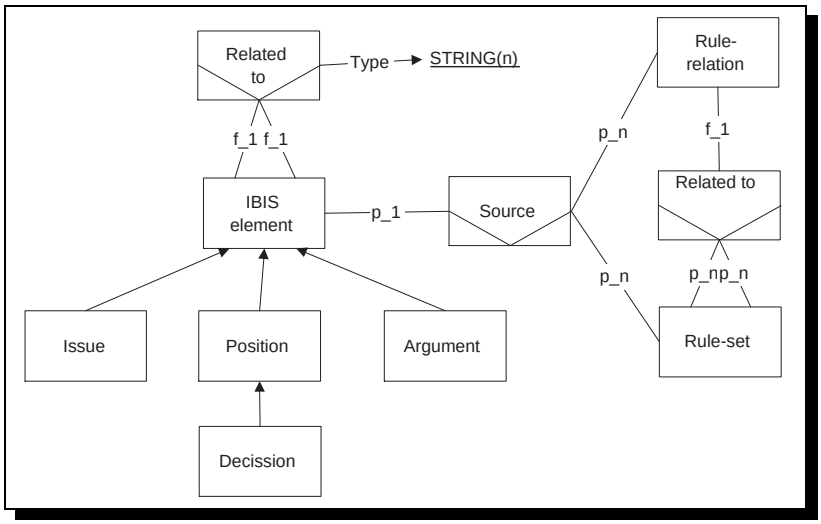

Figure 9.15: Relationship between rule-hierarchies and IBIS

An action or part of action is written ψ_a whereas a state is written ψ_s.
Ignoring the deontic extension for a moment, there are four basic forms of rules.

- ψ : ψ must hold at any tick during execution.
- if ϕ then ψ: If ϕ holds at any tick for some variable substitutions, then ψ must also hold for the same variable substitutions.
- when τ then ψ: If τ holds at any tick for some set of variable substitutions, but did not hold if it had been evaluated at the previous tick ($\neg \bullet \tau \wedge \tau$), then ψ must also hold for the same variable substitutions.
- when τ if ϕ then ψ: If τ and ϕ hold at any tick for some set of variable substitutions, and for which τ did not hold if it had been evaluated at the previous tick, ($\neg \bullet \tau \wedge \tau \wedge \phi$) then ψ must also hold for the same variable substitutions.

An optional 'else' clause may be added to the rules of form 2 and 4, which obeys the following equivalence rules:

if ϕ then ψ_1 else $\psi_2 \equiv$ { if ϕ then ψ_1, if $\neg\phi$ then ψ_2 }

when τ if ϕ then ψ_1 else $\psi_2 \equiv$ {when τ if ϕ then ψ_1,when τ if $\neg\phi$ then ψ_2}

As in Tempora, free variables are universally quantified across the rules, and no free variable should appear only in the scope of a negation. Free variables that appear in an else-expression must also appear in the when-expression. Both rules will have the same deontic operator and role restriction, if this is not indicated specifically. Execution of rules is performed in query-execute cycles where

- For constraint and derivation rules, the consequence is made to hold in the same time as the condition.
- For action rules, the consequence is made to hold in the next execution cycle after the cycle in which the condition holds.

Since the processing capabilities of an actual computer system are always bounded, including the number of rules that can be processed within the duration of one tick, it was proposed in Tempora to operate with two time references [277]:

- The actual real time, denoted t_R.
- The temporal module time, denoted t_M, which is considered as the real time by the temporal modules, but can actually be delayed with respect to t_R.

More precisely, t_M can be defined as follows: After each evaluation cycle, t_M is incremented by one tick. If t_M is in advance of t_R, then the system waits until the latter is equal before beginning the next cycle, else the next cycle begins immediately. In this case, we say that the system is living on "borrowed time". When the load on the system decreases, it will be able to catch up and the two time axes will coincide.

The following predicates will be used in addition to the rule expressions below.

- *borrowed_time*: $t_M < t_R \rightarrow$ *borrowed_time* = true.
- *initiator*(τ, α): The actor α is the initiator of an event τ that triggers a rule.
- *deontic_inconsistency*: A deontic inconsistency has resulted and is reported. At the moment no automatic reconciliation of deontic inconsistencies is suggested. (If a rule of necessity is violated, the resulting action is rejected).
- *recommendation_overruled*: A report of an overruled recommendation is issued. This is no error, but the responsible persons should be notified of the situation.
- *report_discouraged* (ψ_a, α): An action ψ_a that is discouraged is triggered by actor α. This is not an error, but the responsible persons should be notified.
- *security_violation*(ψ_a, α): Reports an attempted security violation, i.e. actor α tries to perform ψ_a without having the authorization to do this.
- *notification*(ψ_a, α): A notification is sent to α about the expected performance of ψ_a.

Derivation rules are always rules of necessity whereas **P** is not used for constraints since they only applies to the CIS (cf. Chapter 7.2). Of the same reason, roles are only specified for action rules.

Action rules

For action rules, we have the following interpretation of the different deontic operators. (For the moment we ignore temporal operators and the exception-hierarchy. ρ = 'the application system' when nothing else is indicated. Constraints and derivation rules are ignored at the moment.). Rules do not change during execution.

- Obligatory action: $\mathcal{E} \wedge \mathbf{O}_\rho\ (\psi_a\ /\ \mathcal{E}) \rightarrow \circ\psi_a$
  For a rule applying to a social actor α, one could perceive the support of the following mechanism:

  $\mathcal{E} \wedge \mathbf{O}_\alpha\ (\psi_a\ /\ \mathcal{E}) \rightarrow$ *notification*(α, ψ_a)

  $\bullet\mathcal{E} \wedge \mathbf{O}_\alpha\ (\psi_a\ /\ \mathcal{E}) \wedge \neg\psi_a \rightarrow$ *deontic_inconsistency*.

  On the other hand, one would often like to have more specific reminder-routines e.g. such as those depicted in Figure A.33.
- Recommended action:

  $\mathcal{E} \wedge \mathbf{R}_\rho\ (\psi_a\ /\ \mathcal{E}) \wedge \neg$ *borrowed_time* $\rightarrow \circ\psi_a$

  $\mathcal{E} \wedge \mathbf{R}_\rho\ (\psi_a\ /\ \mathcal{E}) \wedge$ *borrowed_time* $\rightarrow$ *recommendation_overruled*

  In periods when there is too much to do for the CIS, the use of various deontic operators indicate what actions are the most important to perform. Action rules that are recommended are only executed if we are not living on borrowed time. In this situation we have two choices: Either suspending the recommended action and execute it later when the load decreases, or ignoring the action altogether. In our framework, we use the second semantics. The first situation can be expressed intuitively by using the '$\diamond$' operator. Hence $\mathbf{R}\psi_a \not\equiv \diamond\psi_a$.

The semantics of '$\diamond$' when applied to actions are:

$$\mathcal{E} \wedge \mathbf{O}_\rho\ (\diamond\psi_a\ /\ \mathcal{E}\ ) \wedge \neg\ \textit{borrowed_time} \rightarrow \circ\psi_a$$
$$\mathcal{E} \wedge \mathbf{O}_\rho\ (\diamond\psi_a\ /\ \mathcal{E}) \wedge\ \textit{borrowed_time} \rightarrow (\textit{borrowed_time}\ \mathcal{U}\ (\bullet\ \text{time_is t}) \rightarrow \psi_a\ \text{at t})$$

The action is performed immediately if we are not living on borrowed time, but is suspended until we are no longer living on borrowed time if we are currently living on borrowed time. The action is instantiated at the current time, thus the trigger and condition is not reevaluated when *borrowed_time* becomes false.
In a devtenance project, the low priority indicated by stating that an action is recommended would often result in that it is not implemented in this release at all. On the other hand, it is useful to be able to simulate the behavior of including this.
For a rule applying to a social actor, one could perceive the support of the following mechanism:

$$\mathcal{E} \wedge \mathbf{R}_\alpha\ (\psi_a\ /\ \mathcal{E}) \rightarrow \textit{notification}(\alpha, \psi_a)$$
$$\bullet\mathcal{E} \wedge \mathbf{O}_\alpha\ (\psi_a\ /\ \mathcal{E}) \wedge \neg\psi_a \rightarrow \textit{recommendation_overruled.}$$

- Permitted action:

$$\mathcal{E} \wedge \mathbf{P}_\rho\ (\psi_a\ /\ \mathcal{E}) \wedge \text{initiator}(\tau,\alpha) \wedge ((\alpha \in \rho_s \subset \rho) \vee \alpha \equiv \rho) \rightarrow \circ\psi_a$$
$$\mathcal{E} \wedge \mathbf{P}_\rho\ (\psi_a\ /\ \mathcal{E}) \wedge \text{initiator}(\tau,\alpha) \wedge ((\alpha \notin \rho_s \subset \rho) \wedge \alpha \not\equiv \rho) \rightarrow \textit{security_violation}(\psi_a, \alpha)$$

The semantics of **P** is that the system should support the possibility to perform an action. If no one wants to perform the action, the system does nothing. If someone authorised to perform the action initiates it, it is carried through. If someone not being authorised tries to initiate the action, the security violation is logged. We notice that this is different from the traditional semantics $\mathbf{P}\ \psi \rightarrow \neg\ \mathbf{O}\neg\ \psi$ from deontic logic. As discussed by Maibaum [212] this semantics is not applicable in the case where ψ is an action, since negated actions are not even a correctly formed expression.
In the case of having another rule $\mathbf{P}_{\rho_2}\ (\psi_a\ /\ \mathcal{E})$, where $\rho_2 \neq \rho$, it would be sufficient for the actor to fill only on of these roles.
We will in this overview concentrate on rules applying to individual actors and roles. In case a rule apply to all members of an organizational actor, the rule should be stated to apply to a role "member of OA". Rules can also apply to organizational actor, e.g. it is permitted for a school-class to get in at a museum for a reduced price. This will be similar to above and the initiator of the action to order tickets will be the school-class or someone acting on behalf of the school-class.
- Discouraged action:

$$\mathcal{E} \wedge \mathbf{D}_\rho\ (\psi_a\ /\ \mathcal{E}) \wedge \text{initiator}(\tau,\alpha) \wedge ((\alpha \in \rho_s \subset \rho) \vee \alpha \equiv \rho) \rightarrow \circ\psi_a \wedge \textit{report_discouraged}(\psi_a, \alpha)$$

An action that is discouraged is performed for those that it is discouraged for, but the situation is reported.
If an action that is discouraged is attempted by an unauthorized actor, this is a violation like the above, if there is not another rule which permits the action under the same circumstances for this actor.
- Forbidden action:

$$\mathcal{E} \wedge \mathbf{F}_\rho(\psi_a\ /\ \mathcal{E}) \wedge \text{initiator}(\tau,\alpha) \wedge ((\alpha \in \rho_s \subset \rho) \vee \alpha \equiv \rho) \rightarrow \textit{security_violation}(\psi_a, \alpha)$$

In the case where the same rule is applicable to several roles/actor, which both includes the same actor, and one of these has a permission whereas another one has a prohibition, it is the most specific rule that applies by default, i.e. the rule that applies to the fewest actors. If rule-overuling is indicated though, this might be different.

Constraints

For integrity constraints (which by definition applied only to the application system) the basic rules are:

- $\phi \wedge$ **O** (ψ_s/ϕ) $\Rightarrow$ ψ_s must always hold as long as ϕ holds
- $\phi \wedge$ **R** (ψ_s/ϕ) $\Rightarrow$ ψ_s should hold as long as ϕ holds
- **P**(ψ_s/ϕ) *Not applicable*
- $\phi \wedge$ **D** (ψ_s/ϕ) $\Rightarrow$ ψ_s should not hold as long as ϕ holds
- $\phi \wedge$ **F** (ψ_s/ϕ) $\Rightarrow$ ψ_s must never hold as long as ϕ holds

The reason that the permitted-operator do not apply, is that a permission provides no constraint for the CIS.

When discussing the semantics of the integrity rules at the presence of different kinds of actions, the following short-hands are used:

R1: $\mathbf{O}_\rho$ $(\psi_a/\mathcal{E})$, **R2**: $\mathbf{R}_\rho$ $(\psi_a/\mathcal{E})$, **R3**: $\mathbf{P}_\rho$ $(\psi_a/\mathcal{E})$, **R4**: $\mathbf{D}_\rho$ $(\psi_a/\mathcal{E})$

The set of consequences of the action rule is denoted *consequences(***Rx***)* and the state after the previous tick is regarded to be consistent.

- Obligatory state:

  R1 $\wedge\mathcal{E}\wedge$**O**$(\psi_s/\phi_s)\wedge(\phi\rightarrow\phi_s)\wedge(\neg\psi_s) \subset$ *consequences(***R1***)* $\rightarrow$ *deontic_inconsistency*

  When having two obligations which are inconsistent with each other, this is an inconsistency that we cannot resolve directly.

  R2 $\wedge\mathcal{E}\wedge$**O**$(\psi_s/\phi_s)\wedge(\phi\rightarrow\phi_s)\wedge(\neg\psi_s) \subset$ *consequences(***R2***)* $\rightarrow$ *recommendation_overruled*

  If the constraint rule is obligatory and the action rule is recommended only, the action is discarded.

  R3 $\wedge\mathcal{E}\wedge$**O**$(\psi_s/\phi_s)\wedge(\phi\rightarrow\phi_s)\wedge$ initiator$(\tau,\alpha)\wedge((\alpha \in \rho_s \subset \rho)\vee\alpha \equiv \rho)$ $\wedge(\neg\psi_s) \subset$ *consequences(***R3***)* $\rightarrow$ *deontic_inconsistency*

  The same situation as for the obligatory action above. If the actor is unauthorized, the rule is stopped because of this instead.

  R4 $\wedge\mathcal{E}\wedge$**O**$(\psi_s/\phi_s)\wedge(\phi\rightarrow\phi_s)\wedge$ initiator$(\tau,\alpha)\wedge((\alpha \in \rho_s \subset \rho)\vee\alpha \equiv \rho)$ $\wedge(\neg\psi_s) \subset$ *consequences(***R4***)* $\rightarrow$ *recommendation_overruled*

  The same situation as for the recommended action above. If the action is prohibited, it has already been taken care of by the rules for action rules.

- Recommended state:

  R1 $\wedge\mathcal{E}\wedge$**R**$(\psi_s/\phi_s)\wedge(\phi\rightarrow\phi_s)\wedge$ initiator$(\tau,\alpha)\wedge(\neg\psi_s) \subset$ *consequences(***R1***)* $\rightarrow$ $\circ\psi_a\wedge$ *report_discouraged* (ψ_a, α)

  If we have an obligatory action whose consequences are inconsistent with a recommended constraint, the action is still carried through.

R2 $\wedge\mathcal{E}\wedge$**R**$(\psi_s/\phi_s)\wedge$ $(\phi\rightarrow\phi_s)\wedge\neg$ *borrowed_time* $\wedge$ $(\neg\psi_s)\subset$ *consequences(***R2***)* $\rightarrow$ *recommendation_overruled*

Two inconsistent recommendations gives an inconsistent situation which is parallel to a *deontic_inconsistency*, but which are weaker and thus do not need reconciliation, by rejecting the action.

R3 $\wedge\mathcal{E}\wedge$**R**$(\psi_s/\phi_s)\wedge$ $(\phi\rightarrow\phi_s)\wedge$ initiator(τ,α) $\wedge$ $((\alpha\in\rho_s\subset\rho)\vee\alpha\equiv\rho)$ $\wedge(\neg\psi_s)\subset$ *consequences(***R3***)* $\rightarrow\psi_a\wedge$ *report_discouraged*(ψ_a,α)

Parallel to the case with an obligatory action.

R4 $\wedge\mathcal{E}\wedge$**R**$(\psi_s/\phi_s)\wedge$ $(\phi\rightarrow\phi_s)\wedge$ initiator(τ,α) $\wedge$ $((\alpha\in\rho_s\subset\rho)\vee\alpha\equiv\rho)$ $\wedge(\neg\psi_s)\subset$ *consequences(R4)* $\rightarrow$ *recommendation_overruled*

Parallel to the case with recommended action.

The cases for discouraged and forbidden states are parallel to recommended and obligatory states above.

As illustrated, the deontic operators are restricted to act as meta-operators used under the construction of the set of rules that apply at each tick, and they are not allowed as part of the conditions.

Temporal operators in the scope of deontic operators

There may be temporal operators within the scope of deontic operators in cases where the operators apply to actions of an action-rule. How to treat the different cases is outlined below:

If there are no temporal operators on the action, it is performed immediately and the deontic operators apply as outlined above. When a temporal operator specifies that the action is to be performed some time in the future we will use the following mechanisms for choosing when the action should be performed given the different operators:

- Obligatory actions: The semantics indicated by the temporal operator is applied.
- Recommended actions: This is different for different types of temporal operators:
  - At point in time t:

    $\mathcal{E}\wedge$ **R**$_\rho$ (ψ_a at t / $\mathcal{E}$) $\rightarrow$ ($\neg$ *borrowed_time* at t $\rightarrow$ ψ_a at t+1)
    $\mathcal{E}\wedge$ **R**$_\rho$ (ψ_a at t / $\mathcal{E}$) $\rightarrow$ (*borrowed_time* at t $\rightarrow$ *recommendation_overruled*)

    Thus the situation regarding borrowed_time is assessed at the time the action is meant to take place. If we are living on borrowed time at this time, the recommended action is discarded.
  - During interval [t1,t2]:

    $\mathcal{E}\wedge$ **R**$_\rho$ ($\psi_a\diamond$ during [t1,t2] / $\mathcal{E}$) $\rightarrow$ ($\neg$*borrowed_time* at t $\wedge$ max[tt,t1] $\leq$ t $\leq$ t2 $\wedge$*borrowed_time* $\mathcal{S}$ max[t1,tt] $\rightarrow$ ψ_a at t+1)
    $\mathcal{E}\wedge$ **R**$_\rho$ ($\psi_a\diamond$ during [t1,t2] / $\mathcal{E}$) $\rightarrow$ (*borrowed_time* $\mathcal{S}$ max[t1,tt] $\wedge$ *borrowed_time* $\mathcal{U}$ t2 $\rightarrow$ *recommendation_overrule*)

    'tt' above is the time the rule is triggered. If ψ_a is to be performed within an interval, it will be performed as soon as the system is not living on borrowed time within the interval. If there is not time to do it within the interval, the action will be discarded altogether.
  - Sometime_in_future:

$\mathcal{E} \wedge$ $\mathbf{R}_\rho$ ($\diamond\psi_a$/ $\mathcal{E}$) $\wedge\neg$ *borrowed_time* $\rightarrow$ $\mathbf{R}_\rho$ (ψ_a)
$\mathcal{E} \wedge$ $\mathbf{R}_\rho$ ($\diamond\psi_a$ / $\mathcal{E}$)$\wedge$ *borrowed_time* $\rightarrow$ (*borrowed_time* $\mathcal{U}$ ($\bullet$ time_is t) $\rightarrow$ $\mathbf{R}_\rho$ (ψ_a at t))

Since it is not essential that ψ_a is performed immediately as it usually is for recommendations this situation is treated as an obligation with the exception that it can potentially be overruled by another obligation when the rule fire.

- Permitted actions
  The authorization is checked as indicated earlier when the action is performed.
  Also *discouraged* actions are parallel to the earlier discussion, whereas having temporal specifications for rules having *forbidden* as the deontic operator only will enforce the prohibition for the time indicated.

Exception hierarchy

The following run-time semantics for the overrule and suspend relationships are used.

- Overrules: This can be translated in the following way:

  R1: ∇_ρ (ψ_1 / $\neg \bullet \tau \wedge \tau \wedge \phi_1$) , **R2**: ∇_ρ (ψ_2 / $\neg \bullet \tau \wedge \tau \wedge \phi_2$), **R1** *overrules* **R2**
  $\Rightarrow$
  R1: ∇_ρ (ψ_1 / $\neg \bullet \tau \wedge \tau \wedge \phi_1$) , **R2**: ∇_ρ (ψ_2 / $\neg \bullet \tau \wedge \tau \wedge \neg\phi_1 \wedge \phi_2$)

  That this is in fact a short-hand is not so obvious here, but consider the situation where ϕ_1 is complex. If **R3:** $\nabla(\phi_1/\phi_2)$ is a constraint, R2 will never be activated.
- Suspends can be translated in the following way:

  R1: ∇_ρ (ψ_1 / $\neg \bullet \tau \wedge \tau \wedge \phi_1$) , **R2**: ∇_ρ (ψ_2 / $\neg \bullet \tau \wedge \tau \wedge \phi_2$) **R1**, *suspends* **R2**
  $\Rightarrow$
  R1: ∇_ρ (ψ_1 / $\neg \bullet \tau \wedge \tau \wedge \phi_1$) , **R2a**: ∇_ρ (ψ_2 / $\neg \bullet \tau \wedge \tau \wedge \neg\phi_1 \wedge \phi_2$)
  R2b: ∇_ρ (ψ_2 at t / $\neg \bullet \tau \wedge \tau \wedge (\phi_1 \mathcal{U}$ t) $\wedge \phi_2$)

Figure 9.16 illustrates different situations where role overruling and the principle of specificity can come into effect. In case of conflict, the selection order would be R4, R1, R2, R3.

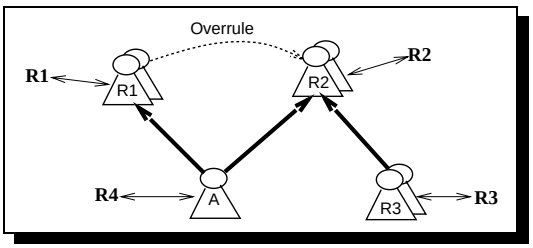

Figure 9.16: Over-ruling mechanisms based on the actor model

Execution cycle

To bring the approach together, this section present the execution cycle incorporating the deontic operators. It is somewhat different from the current implementation of the Tempora prototype [307], which is not handling non-determinism even if the principles for how to include this

are specified. Since the main parts of the cycle are illustrated formally above, the overview is in natural language for the case of brevity. As in Tempora it should be possible to specify transactions including several action rules. If not part of a larger transaction, the single rule is regarded as a transaction.

- 1. t_M and t_R are compared to see if the application system is living on borrowed time.
- 2. A primary set of rules $\mathcal{R}_{\mathcal{P}}$ is constructed utilizing information about previously executed rules, and necessitation-relationships between rules.
- 3. The condition of every rule in $\mathcal{R}_{\mathcal{P}}$ is evaluated, and for all successful evaluations, the rule and its accompanying transaction is put into the rule-set for this tick $\mathcal{R}_{\mathcal{I}}$. In addition, previously rolled-back and suspended rules that now apply will be put into this set.
- 4. $\mathcal{R}_{\mathcal{I}}$ are checked for the following:
  - A recommendation will be discarded if the system is living on borrowed time.
  - A rule that is prefixed by '$\diamond$' or another temporal expression with similar non-determinism will be suspended if the system is living on borrowed time.
  - A rule that is recommended and that is supposed to be performed within a given time interval, is suspended if the time interval is not reached yet or if the time interval is not finished and the system is living on borrowed time. The rule is discarded if the time-interval has ended and the system is living on borrowed time.
  - If a rule is initiated by an actor not authorized to perform the action, the rule will be discarded.
  - If a rule is overruled by another rule that is not overruled by a third rule, the rule is discarded given that all these rules are enabled.
  - Likewise for suspensions, but the action is to be taken into consideration at a later point in time. Both for overrule and suspend-relations we might avoid the traversing of the exception hierarchies during run time using the rewrite rules indicated above. Similarly might be done for the over-ruling mechanism based on the actor model [54].

  The consequences of the resulting rules will then be collected into the action set of the tick $\mathcal{R}_{\mathcal{AC}}$.
- 5. $\mathcal{R}_{\mathcal{AC}}$ is checked to see if it is self-contradictory, contradictions being resolved if possible according to rules above. Rules that are necessarily inconsistent are discarded whereas those that are deontically inconsistent are left to human decision.
- 6. The transactions are executed, causing external events to occur, changing databases, or scheduling new actions for future times.
- 7. While $t_M < t_R$ then wait, else move to the next tick on the event time axis, making available any external input which arrived during the evaluation of the previous tick.

9.3.3 Extension of Consistency Checking

Both the consistency-checking of rule-bases and of process models are influenced by our suggestions, and we will briefly treat both cases below.

Deontic rules

The following axioms relating to the use of deontic operators within the rules can be used during analysis to reduce the rule-base to one with two modalities, and by that enabling the use of traditional consistency-checking techniques. The following mechanisms are not used during runtime. The axioms are inspired by the work of Jones and Pörn [155, 156] and Maibaum [212] in

addition to standard deontic logic [321]. It is distinguished between action rules and constraints.

Action rules

- **A1**: $\mathbf{O}_\rho\ (\psi_a\ /\ \mathcal{E}) \Rightarrow \mathbf{R}_\rho\ (\psi_a\ /\mathcal{E})$
  An action that is obligatory in a given situation is recommended in the same situation.
- **A2**: $\mathbf{R}_\rho\ (\psi_a\ /\ \mathcal{E}) \Rightarrow \mathbf{P}_\rho\ (\psi_a\ /\mathcal{E})$
  An action that is recommended in a given situation is permitted in the same situation.
- **A3**: $\mathbf{F}_\rho\ (\psi_a\ /\ \mathcal{E}) \Rightarrow \mathbf{D}_\rho\ (\psi_a\ /\ \mathcal{E})$
  An action that is forbidden in a given situation is discouraged in the same situation.
- **A4**: $\mathbf{P}_{\rho_1}\ (\psi_a\ /\ \mathcal{E}) \Rightarrow \mathbf{F}_{\rho_2}\ (\psi_a\ /\ \mathcal{E})\ ,\ \rho_2 \not\subset \rho_1$
  If an action is permitted to be executed by an actor in the role ρ in the given situation, then it is not allowed for an actor of any role that is not a subset of the permitted role to initiate the action.

Using the above axioms and transitivity, all actions can be reduced to permitted and discouraged actions.

Constraints

- **A5**: $\mathbf{O}(\psi_s\ /\ \phi) \Rightarrow \mathbf{R}(\psi_s\ /\ \phi)$
  A state that is obligatory in a situation is recommended in the same situation.
- **A6**: $\mathbf{D}(\psi_s\ /\ \phi) \Rightarrow \mathbf{R}(\neg\psi_s\ /\ \phi)$
  If a state is discouraged in a given situation, then the opposite state is recommended in the same situation.
- **A7**: $\mathbf{F}(\psi_s\ /\ \phi) \Rightarrow \mathbf{O}(\neg\psi_s\ /\ \phi)$
  A state that is forbidden under a given situation is obligatory to avoid.
- **A8**: $\mathbf{R}(\psi_s\ /\ \phi) \Rightarrow \mathbf{P}(\psi_s\ /\ \phi)$
  A state that is recommended under a given situation is permitted in the same situation.

Using these axioms and transitivity all constraints can be reduced to permissions.

Extended use of actors in PPM-diagrams

It might now be possible to decompose a process such that the input-flow is first sent to an actor. In this case, it is not straightforward to use all the earlier proposed techniques for consistency checking [340]. By specifying rules or I/O-matrices for actors and roles, these techniques can still be supported.

The rules outlined in Section 9.1 should also be included in the consistency-checking support of a tool supporting the approach.

9.3.4 Driving Questions

A technique that can be used during modeling in a conceptual framework integrating several conceptual modeling languages is the use of *driving questions*. The overview here is partly based on work on Tempora [307], but extended to take the whole conceptual framework into account. Different questions will be more or less applicable according to the subset of the modeling languages that are used in the different modeling activities as described in Chapter 10. One should also be aware of not only pursuing the avenues for modeling that open up based on the use of this kind of questions, since this can hinder the development of models that are not so easily represented in the more specialized parts of the languages.

ONER-modeling

This can be performed in isolation according to guidelines for semantic data modeling. Driving questions for interaction with the other languages are given below.
Given a role in an actor-model or process-model:

- Is the structural aspects of the role represented in ONER? One might either find the role represented as an entity-class, or as the role an entity-class has in a relationship-class.

Given a process model:

- For a store: Is the contents of the store represented in an ONER-model?
- For a flow: Is the data-items traveling on this flow represented in the ONER-model?
- For a process: What data do the process use?

Given a rule:

- Is all structural components that the rule refer to found in the ONER-model?

Process model

Process modeling in isolation is similar to DFD-modeling, with the extensions indicated above. When using the speech-act modeling extension though, it should be closer to action-workflow, using conversations as the main structuring principle.

Driving questions in connection with the use of several languages are given below:
Given a rule:

- Does the rule indicate a process that is not currently found in the process model?

Given an actor or role:

- Do the actor/role have a supporting or communicating position in the process model?
- Do an existing actor/role support a process or a store that is not yet depicted?

Given an entity or scenario:

- How do instances of a class come into existence? When do they cease to exist?
- When and how are relationships established and removed?
- Is the entity being processed by any process?
- Is the entity being stored in any existing store?
- Is the entity or part of the entity-class transported as items on any flow?

Given an illocutionary act:

- Is there flows in this (or other) process models for all illocutionary acts of a given conversation?

Rule modeling

The participants can be asked general questions such as if there are any specific types of restrictions that are valid for a particular area, or if there are any specific policies or types of guidelines that are used in the day to day running of the organization.
Given social actors and roles:

- What rules apply to a given role, or to a given actor (e.g. yourself) in a given situation?
- Does the rule applying to a super-role also apply to the sub-role?
- Does all rules applying to the sub-role also apply to the super-role?
- Does a rule applying to an actor also apply to any of the roles that the actor fill?
- Do all rules applying to a role, apply to all the actors filling the role?

Given a certain entity or relationship class:

- How do instances of a class come into existence? When do they cease to exist?

- For a rule valid for a subclass, is it also valid for the superclass?
- For a rule valid for a class, is it also valid for the subclasses?
- For a rule valid for a class, is it also valid for sibling classes?
- When and how are relationships established and removed?

Given a certain process:

- What is triggering a process?
- What rules applies to the internal functioning of the process?
- What are the different cases and what actions should be performed for each case? I/O-matrices may be useful here to structure the answers before they are represented as rules.

Given an illocutionary act:

- Is there any rules being implied by the performance of the illocutionary act that is not yet represented?

Given a timer:

- What is the rule resulting in the sending of each outflow?
- Is all inflows and offlows covered by rules?

Given a rule:

- Is this rule internally or externally institutionalized? If it is externally institutionalized, which internal rules need this to be in force to permit the internal rule?
- Do the rule depend on the existence of an externally institutionalized rule?
- What rules does the rule contribute to fulfilling? How else is this rule realized?
- How can this rule be realized?
- Are there alternative ways of realizing the rule than what is already indicated?
- Are there rules that work against this rule?
- Do the rule use terms which need to be defined by other rules?
- Is the rule a more restricted version of another rule?
- Are there any major exceptions for the rule?

Actor-modeling

Given an actor or role:

- For an organizational actor, is all its parts modeled?
- For any actor, is all relevant actors that it is a part of modeled?
- For a role, is all actors that have this role modeled?
- For a role, is all the institutionalizing actors of the roles given.
- For a role, is all relevant sub-roles of the role given?
- For a role, is this a sub-role of another role?
- For an actor, is all of the relevant roles of this actor included?
- For a support-relationship, do the supporter support in a given role? On behalf of a given actor? Is the supportee supported in a given role? On behalf of a given actor?
- For a communication, do the sender communicate in a given role? On behalf of a given actor? Is the receiver receiving this item in a given role? On behalf of another actor?
- Is an actor part of another actor in a given role? On behalf of a given actor?
- Is there a power relationship between this role/actor and another?

Given a process model:

- Is there an actor or role either communicating or supporting, that is not included in the actor-model?

Given an entity-class:

- Do the entity-class correspond to a role that is not included in the actor-model?
- Is there specializations between entity-classes that are also represented as roles, that are not indicated as super-roles?

Given a rule:

- Does the rule apply to a role or an actor that is not included in the actor-model?
- Is the rule institutionalized by an actor that is not included in the actor-model?

Additional metrics for completeness and validity

When discussing the use of specific modeling languages, it is possible to come up with more specific metrics than could be done in Chapter 6 when discussing this in general terms.

We will here, inspired by the suggestions for metrics for requirements traceability and completeness [58] and the above come up with some proposals for metrics in connection to the use of rule-hierarchies.

- Validity: The number of rules that are not necessitated, obligated or recommended by another rule. A ratio of this compared to the total number of rules might also be interesting. One will reach a point where one is not longer able to come up with higher goals motivating for a rule [339]. These goals should be indicated specifically and not be counted when calculating the metric-value.
- Completeness: The number of rules that are necessitated, obligated or recommended based on a given rule. Three cases are of special interest:
  - 0 links: The work in this area is to be completed. If the rule is a detailed rule for the processing of a process or giving explicit constraints on the ONER-model, and is thus not meant to be further specialized, the rule is not regarded. In the case this rule belongs to an area that it is decided to not look into further for the moment, the same applies.
  - 1 or 2 links: This might indicate that the analysis of the rule has been superficial and it needs to be worked on further.
  - The rule has a high downward degree: This may mean that the higher level rule itself is too general, and should be split into several high-level rules before being linked to the next level.
- Unresolved issues: The number of downward or-nodes, discouragement, prohibitions, and exclusions that are not addressed.

Similar metrics could be devised for the overall approach, but this is not discussed in detail here.

9.4 Chapter Summary

We have in this chapter specified a possible extension of the conceptual modeling languages in PPP on two areas:

- Explicit modeling of actors, roles, and actor interaction, and relating these phenomena to existing modeling languages in PPP.
- Explicit modeling of rules, and relating rules to the other modeling languages in PPP and to other rules.

We have also indicated how some existing techniques in PPP for supporting the construction of conceptual models of high quality can be extended to take our conceptual extensions into account. Additional extensions are described briefly in Appendix D.

The overall conceptual framework is with this covering the following perspectives:

- ONER covers the structural perspective.
- PPM covers both the functional and the behavioral perspective. PLD and matrices are used to model detailed behavior.
- DRL and the rule-relationships covers the rule perspective and enhances the modeling of the structural, functional, and behavioral perspective.
- The actor and role perspective is addressed through the extensions of the actor modeling language.
- The communication perspective is considered by the addition of illocutionary points, propositional contents, and dominant claim of items sent on a flow. Illocutionary acts can further be linked to rules.
- The object-oriented perspective is partly covered by the use of ONER and the actor-modeling language. Also the UID-language applies object-oriented mechanisms.

We will in the next chapter look upon how the extended conceptual framework, including the modeling of structure, functions, behavior, rules, communication, actors, and roles can be applied in system devtenance. In Chapter 11 the approach will be evaluated.

Chapter 10

Conceptual Modeling in Systems Devtenance

In this chapter we will present a more detailed way of applying the conceptual framework as described in Chapter 9 and Chapter 7.4 in the methodological framework presented in Chapter 8. To illustrate the use of conceptual models within the methodology, we will use examples from the case-studies described in Appendix A.

The chapter contains the following main parts:

- A short discussion on the overall introduction of system devtenance in an organization is given partly based on results from Chapter 3. Aspects such as organizational and actor support modeling, portfolio modeling, COIS planning, and types of devtenance projects are discussed briefly.
- Description of general process heuristics which are linked up to the discussion in Chapter 6 on quality of conceptual models. A modeling task is regarded as being in one of the following states: Preparation, expansion, consolidation, and suspension. The heuristics give guidelines for the transition between states.
- Guidelines for using the conceptual framework described in Chapter 9 and Chapter 7.4 in system devtenance projects. This is divided into three parts:
  - Conceptual modeling in devtenance projects with no preliminary conceptual descriptions.
  - Application of conceptual models in usage support.
  - Conceptual modeling based on existing conceptual descriptions.

  The first part will include the main discussion, and is ordered according to the four kinds of models discussed in Chapter 6 i.e. $\mathcal{M}(EIS)$, $\mathcal{M}(FIS)$, $\mathcal{M}(FCIS)$, and the implemented CIS. Examples of the use of different techniques, such as CATWOE-analysis, actor modeling, process modeling, model integration, and the use of rule-hierarchies are interspersed in the description.

It is important to appreciate that what is presented in this chapter is only one of several possible approaches to system devtenace. It should not be looked upon as a suggestion for an ultimate methodology, but rather as guidelines. We focus primarily on the construction of conceptual models.

10.1 Introducing System Devtenance in an Organization

The starting point from which a systems devtenance methodology is to be introduced into an organization will be varying. Whereas some organizations already are familiar with CASE-tools and have conceptual descriptions in semi-formal or formal languages of many of their existing systems, other organizations are not familiar with conceptual modeling. As indicated in Chapter 3, it appears that it is primarily larger organizations that have introduced CASE-technology, including conceptual modeling such as process and data modeling, but even very few of these have a situation where a large part of the portfolio is currently supported by the CASE-tools being used.

As indicated in Chapter 3, 40% of the organizations claimed to perform maintenance in a project-like manner, and very few had a separate organization of development and maintenance, thus many organizations should be able to adopt the overall methodological framework without too large changes in the current organization of CIS-support.

For most organizations, the incremental strategy presented below is believed to be useful when working towards the support of a system devtenace methodology. What must be noted is that it will be a long term effort to transfer from an existing methodology to the use of system devtenance, and that for many it will not be feasible to transfer large parts of the current portfolio into a conceptual framework, both of technical and economical reasons. Many application systems have a comparatively small user-group, or they are not used extensively by their users thus the overhead of reverse engineering many applications will often be higher than the benefit. A more technical problem arises because of the large number of supportive software and hardware that is in used. As seen in our survey, it is not unusual for a large organization to have several technical architectures and applications implemented in several different programming languages. Since this also differs widely between different organizations, it is not feasible in the short run to be able to support reverse engineering from all these platforms to the conceptual framework. Similarly, this multitude of supporting software will be around for a long time and further evolve, thus one might want to have code-generation facilities for a large number of platforms. A partly solution to this is the use of so-called middelware technology such as API-LINK, CORBA, and COM/OLE which is supposed to improve interoperability between different hardware and supportive software actors. It is beyond the scope of this thesis to discuss middelware in any detail.

A suggestion for an overall approach to introduce system devtenance in an organization is described below.

Develop an organizational model: The formal organizational structure is one source to the creation of this, but will not cover all potentially interesting social actors. The formal organizational structure will also suggest many of the internal roles of the organization. Another way of identifying social actors in organizations which uses electronic mail, is to look on the mail-groups that have been created, indicating groups of persons that have or have earlier had the need to communicate with each other. In organizations where electronic mail is used extensively, maintenance of the organizational model and mail-aliases can be integrated, thus the model can be a help for the front-end support. As an example of an organizational model, the overall organizational model for the IS-group is given in Figure 10.1. This shows the breakdown into individual actors of three of the organizational actors depicted in Figure 9.2, and their roles within IDT and their groups.

Develop overall technical actor interaction model: Start with creating an overall actor-model based on existing application systems in the organization and how these interact with other computational actors both within and outside the organization. This is knowledge which exist mostly

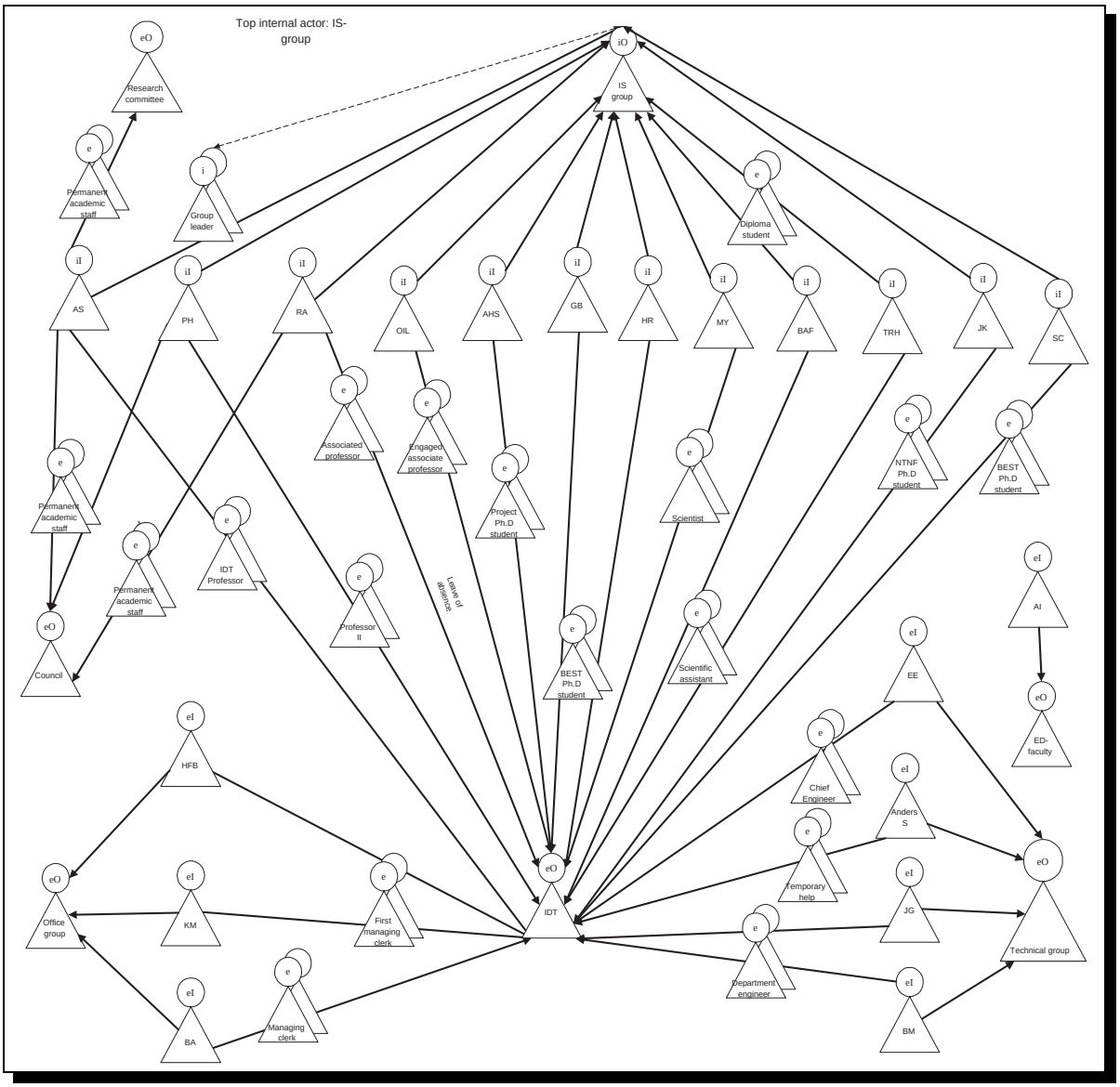

Figure 10.1: Overall organizational model of IS-group

within an IS-department that is often not collected in one place, thus the creation of such models can be of immediate use in the maintenance of the existing portfolio due to the high degree of interconnectedness among major application systems in many organizations. On the other hand it is not a big undertaking, especially since this data can be recorded incrementally. In this way, a high-level framework to support the further evolution of the portfolio can be developed with comparatively little effort. Figure 10.2 indicates how a set of applicative actors being part of a bibliography-system is integrated. This system is further described in Appendix A.

Develop actor support diagrams: When having both an organizational model and technical actor interaction diagrams, one can extend these by developing actor support diagrams, i.e. indicating which actors are supported by other actors in given roles. Technical actors supporting other technical actors might already be found in the actor interaction diagram. To get a first cut of this

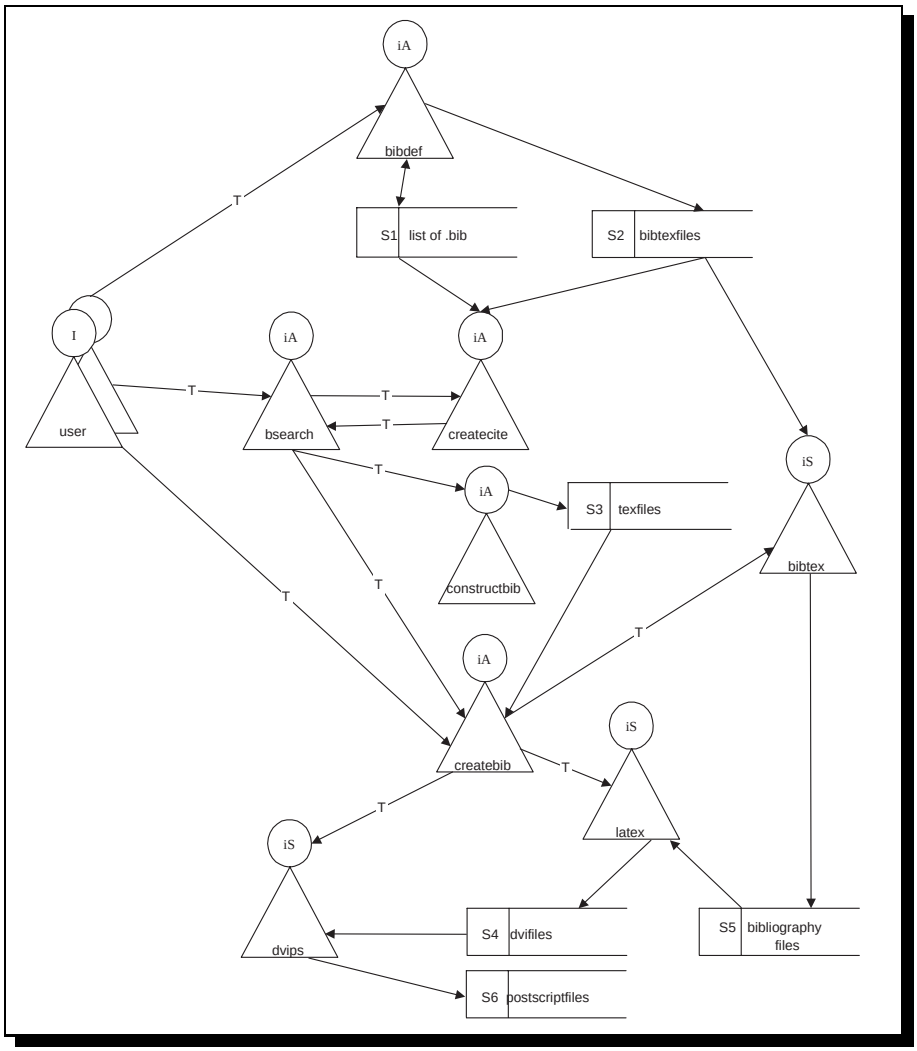

Figure 10.2: Example of technical actor integration diagram

one might use a survey-technique to investigate who is getting support from whom, and follow up the survey with a clarification round. Some of this kind of information might also already be available in the front-end part of the IS-department. Figure 10.3 indicates how the individuals of the IS-group of IDT is supported by other social actors.

Figure 10.4 indicates how members of the IS-group are supported by a selected technical actor, in this case Microsoft Word.

Perform COIS planning: COIS planning, establishing an overall strategy for CIS-support of the organization is generally believed to be beneficial. As part of this work, one might want to create so-called enterprise models. These models are of a strategic nature and its scope can be the whole organization. This provides a model based on the local reality of high-level planning staff, and can provide a basis for a dialogue with user groups in more traditional devtenance projects, leading to organizational learning. Thus an enterprise-wide model is not in the first place to act as a starting point for a corporate database being developed in a top-down manner, but as one of several views to the organizational reality.

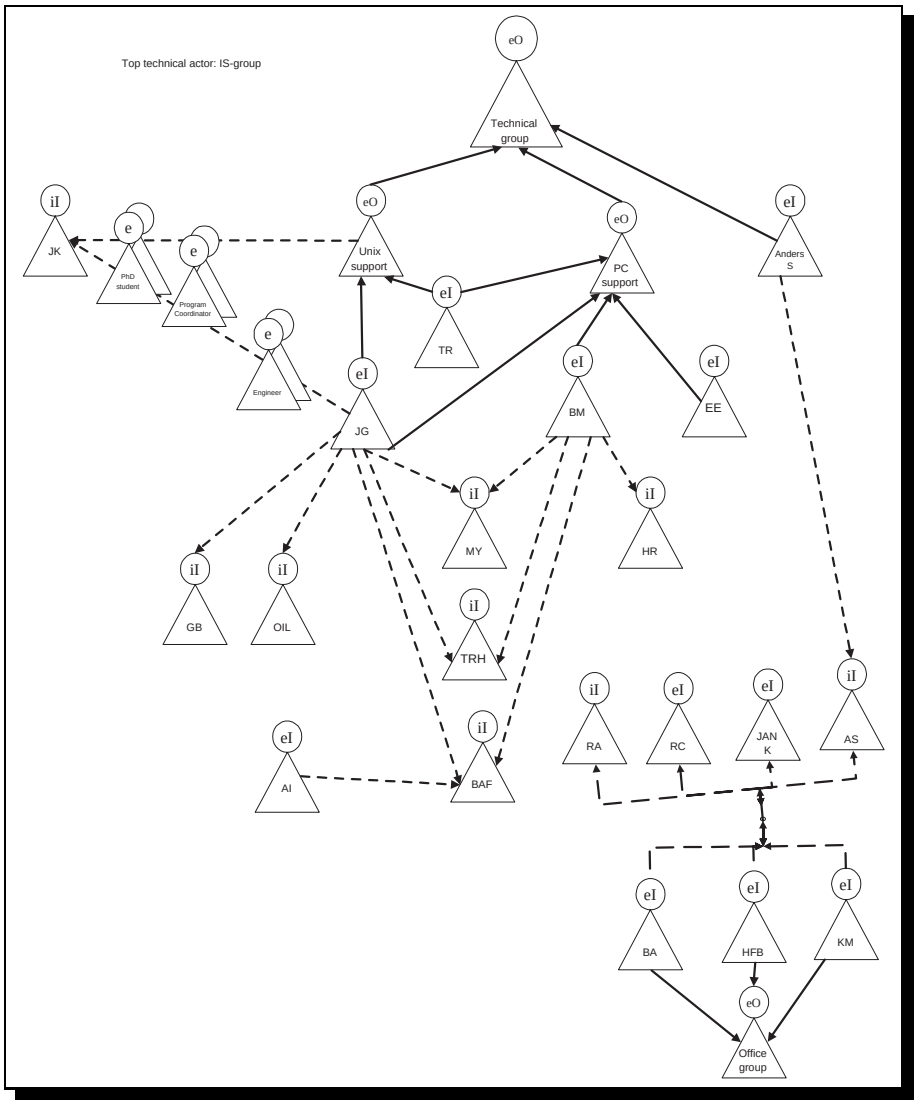

Figure 10.3: Example of social actor support

The enterprise-wide models can be developed in much the same manner as is used when modeling the perceived current situation as described in Section 10.2.2.

Perform individual devtenance projects: As part of devtenance projects, individual models are created, and the models being implemented are connected to an applicative actor.

Devtenance projects can crudely be divided into the following categories:

- 1. New development. A CIS is to be made within a functional area which are not previously supported in the organization by computers.
- 2. Maintenance of an existing application system, having a conceptual model of the system as a starting point.
- 3. Maintenance of an existing application system, without having a conceptual model of the system. This might include:
- 4. Reverse engineering of an existing application system to establish a conceptual model which can then be the starting point for further work.

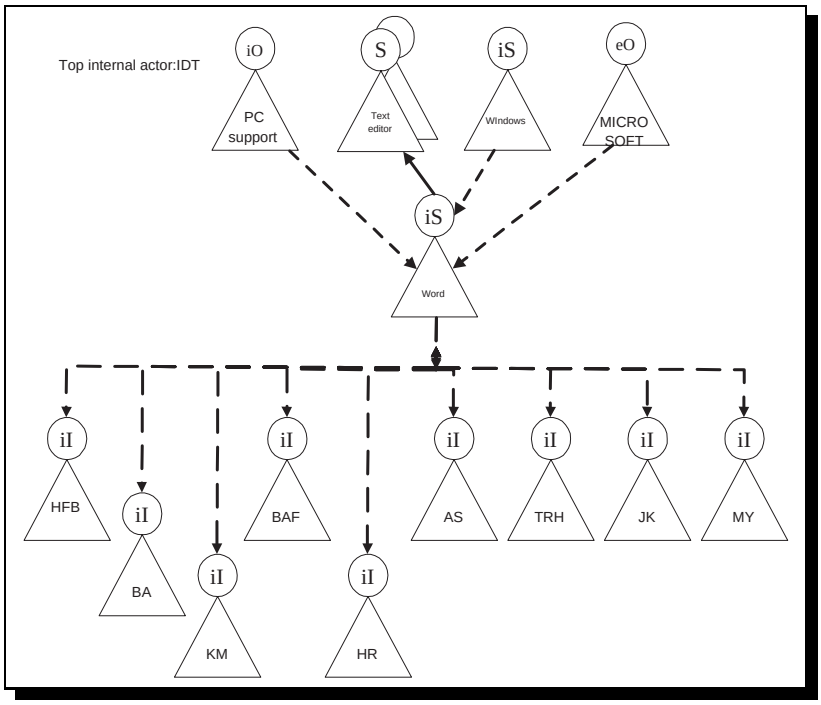

Figure 10.4: Example of technical actor support

- 5. Replacement of one or more existing application system, having a conceptual model of the system(s) as a starting point.
- 6. Replacement of one or more existing application system, not having a conceptual model of the system(s). This might also include reverse engineering to create such model.

We will in this chapter concentrate on projects of type 1, 2 and partly 5. Reverse engineering of existing systems into our conceptual framework is a topic of the size of a separate thesis effort, thus a detailed discussion of this subject is regarded as beyond the scope of the thesis. To support projects of type 3 one can look to methodologies for maintenance (e.g. CONFORM) for inspiration.

Another important aspect of CIS-support is the support of application systems usage, including the handling of COISIRs. This will also be discussed briefly, especially the aspects which can be supported by conceptual models and the links between conceptual models in the specification base and the application system in production.

10.2 Conceptual Modeling in Systems Devtenance

There is obviously much more to devtenance of CISs than conceptual modeling. On the other hand, conceptual modeling is looked upon as an important technique for reducing the gap between the user-communities local reality and computing technology supporting the social construction of the technology. Conceptual modeling is also looked upon as important for supporting both generative and compositional reuse both on a project and portfolio level over the life time of the system. This do not mean that we are not aware of other techniques for specifying requirements such as ethnographic techniques [106]. Where appropriate, also other parts of system devtenance are mentioned including their links to conceptual modeling, but these aspects will

not be discussed in detail. This indicates that our suggested guidelines for a methodology is not in any sense complete. We are for instance not discussing project establishment which could include many techniques from organizational development. One specific technique found in organizational development literature that has influenced the description given below, is the search conference technique [85, 274]. This technique applies similar principles to participation as we have outlined in Chapter 8. The technique has also been used earlier within application development in Norway [309].

The approach to conceptual modeling that we will suggest have some similarities with the Tempora-approach presented in Chapter 7.2. The modeling methodology is updated, including the use of techniques developed in PPP as reported in Chapter 7.4 to enhance the quality of the models and focusing on the social construction process.

As described in Chapter 8, development of conceptual models is divided into several tasks, based on differences in the modeling domain:

- Development of conceptual models of the perceived current situation:
  The behavior, structure, rules, and actors in the problem domain including both computerized and manual data processing in the organization are described and analyzed. This is a description of the current organizational reality as it is internalized by the participants. A learning process about the current situation is meant to take place among all participants, in particular among future users and the developers of the system. This "phase" is parallel to the first or second group-work in a search conference, where the current situation and the passive future-image are described respectively.
- Development of conceptual models of an perceived improved situation:
  This is also performed not taking a CIS solution into account.
- Development of conceptual model of the future CIS:
  The behavior and structure of an application system are described and analyzed. Everything needed for an executable specification should be provided. This also includes the parallel and potentially intertwined development of a user-interface description. The models are late in this phase extended to include detailed design and computational semantics are specified.
- Implementation:
  Based on the selected supportive actors, this is a more or less automatic translation of the design into an application system.

As already mentioned, different project models can be overlaid the proposed approach. We will thus not present the methodology as a set of interconnected tasks although several such models should be possible to overlay the tasks described, making the approach more suitable for project management. Project management as such is regarded as being beyond the scope of this thesis. What we will present, is a set of general process heuristics linking the different modeling efforts up to the quality framework presented in Chapter 6. This overview is based on work performed in cooperation with Sindre [287]. After this we will look in more detail upon a possible application of our conceptual framework in more detail for the different modeling efforts individually.

10.2.1 Process Heuristics in Conceptual Modeling

Heuristics based on data which can be collected during modeling can be used to:

- Guide the actions of the current project.
- Guide the evolution of the organization's devtenance methodology.

We will here focus on the first subject.

Overall idea of the process

In [199] the modeling process is vaguely described as consisting of cycles of expansion and consolidation of $\mathcal{M}$. In the following we will elaborate more on this idea, to prepare the ground for a discussion of process heuristics. Our idea of the process is illustrated as a state machine in Figure 10.5. The diagram can be explained as follows:

- **P - preparation:** In this state, the organization is performing actions in preparation of the modeling itself, e.g. selection of participants, training, and planning.
- **E - expansion:** Model statements are given, hence $\mathcal{M}$ is growing. During expansion, statements may be made more or less uncritically, i.e. thorough validation is not undertaken, and there might be errors introduced in $\mathcal{M}$. Still, as long as some valid statements are made, $\mathcal{M}$'s degree of *completeness* will grow.
- **C - consolidation:** The model statements (especially those captured in the previous expansion phase) are consolidated with respect to *validity*, *comprehension*, and *agreement*, as defined in Chapter 6.
- **S – suspension:** The modeling activity is suspended. There may be several reasons for suspension. The model may have been agreed upon and "frozen" (e.g. to start the next modeling phase), or the project may have been aborted.

This diagram consists of an inner cycle of expansion and consolidation, and an outer cycle including preparation and suspension. The starting state has been defined as **S**, i.e. before you do anything, you are in a state of suspension. The fact that there is no accepting state reflects the view that a COIS is never finished.

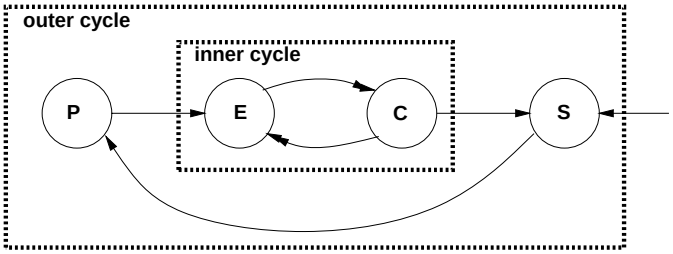

Figure 10.5: The SPEC-cycle for modeling

Considering the entire modeling effort, there will be a lot of work in parallel by various participants. Hence, it is generally impossible to observe the simple state-changes of Figure 10.5 for the model as a whole. It applies to smaller parts, the entire process then being composed of multiple such cycles.

The focus of the heuristics to be discussed here will mostly be on the inner cycle, in particular the switching between **E** and **C**. This switch can be based on the following:

- *Resource limit:* You are supposed to use a certain amount of time or manpower for **E**, then go to **C**, and vice versa, and similarly for **E** and **C** together vs. **S**.
- *Chunk size:* The number of statements made at one visit in **E**. When this size has been reached, there may be a policy to switch to **C**.
- *Progress:* You observe the progress made at **E** or **C** and switch when this has fallen below a certain threshold. The progress *will* decrease when the process has been in the same state for a while because of a phenomenon that may be called *exhaustion:* the most evident statements will be stated first, and the most evident errors found first. Moreover, staying

too long in **E** will yield a big chunk, for which incomprehension and disagreement is likely to hinder further growth of the model.

The three above strategies will be combined to make a practical process.

Gathering of process data

To start with, we will identify what data should be delivered to enable the heuristics. Then we will continue by briefly discussing how this can be obtained.

To determine the progress of modeling and the extent to which feasible quality has been reached, it is useful to know the point estimates for the quality goals, and their corresponding ratios (vs. resource consumption). In addition, model size has been included because it is important in the management of expansion, and model value because it is important in considerations about feasibility.

- perceived validity (PV), and ratio of perceived validity (PVR)
- perceived completeness (PC), and (PCR)
- perceived pragmatic quality (PP), and (PPR)
- perceived social quality (PS), and (PSR)
- perceived knowledge quality (PK), and (PKR)
- model size (MS), and (MSR)
- model value (MV), and (MVR)

The model size should be possible to obtain automatically. For the other data, one need to register:

- For all parts of the model, which have been perceived as complete (within their scope), which have been perceived as incomplete, and how big is this incompletion estimated to be.
- For all statements, which have been acknowledged as comprehended, which have been turned out to be incomprehensible, and which have not yet been checked.
- For all statements, which have been agreed upon and by whom, which have been disagreed upon and by whom, and which have not yet been checked.
- For each activity (here: each visit at **E** or **C**), how much resources are spent.
- For each activity, what is the perceived value increment to the model.

Some of these are easier to obtain than others. The most complicated are perceived completeness and model value. Even if these are dropped, it will be possible to provide some useful heuristics, as will be shown below. The phrase "for all statements" may give the impression that the above requires an enormous registration work. However, this need not be the case. E.g. discussing agreement of an ONER-model, it should be easy to implement functionality for quickly selecting larger parts of the diagram to mark it as agreed, open, or disagreed upon. Depending on the modeling situation (i.e. which languages that are used, and which parts of the languages that are used) one can get support from a CASE-tool to assess several of these based on the knowledge of the modeling languages and metrics as indicated in Chapter 9.3.4. Tool support for this is further indicated in Appendix D.4.

Heuristics to guide the current process

Heuristics will be presented as observed symptoms and possible actions. The list is not supposed to be exhaustive, and as one symptom may have several causes, there are several possible actions for each symptom. Heuristics must be evaluated by the participants in any specific case.

Expansion heuristics

Symptom E1: Resource consumption $\sim$ limit.

Symptom E2: MS increment $\sim$ recommended chunk size.

Action (E1 or E2) :

Switch to consolidation. For E2, if the recommended size was reached very easily, it might also be that the problem is simpler than assumed, so that it can be interesting to increase the recommended chunk size.

Symptom E3: MSR $<$ min.threshold (expansion is getting unproductive).

Symptom E4: MVR < 1 (growth of model value is less than resources being spent, i.e. work currently being done is perceived to yield deficit).

Actions (E3 or E4) :

- Switch to consolidation (if the problem is due to significant incomprehension or disagreement). If the chunk is well within the recommended size, it may also be sensible to lower the recommended chunk size.
- Switch to other techniques (if the problem is due to exhaustive use of some techniques and there are others which can be tried).
- Involve new participants (if the problem is due to exhaustive use of some participants and there are others which is perceived to possess relevant knowledge).

As shown here, an observed symptom from the collected information will not define a unique action; there has to be further considerations by the participants.

Consolidation heuristics

Consolidation heuristics are more complex than expansion heuristics, since there are more goals and measures involved. To save space, we will avoid listing the most obvious heuristics. Hence, it is sensible to address pragmatic quality before validity, completeness, and agreement because comprehension of the model is necessary to achieve anything else with some certainty. Further, it is sensible to address validity and agreement before completeness. Guidance for this sequencing can be done by heuristics investigating the values PP, PV, PS, PC. This will not be discussed below. Instead we will look at symptoms indicating problems with the consolidation being done.

Symptom C1: Resource consumption $\sim$ limit.

Action: Switch to expansion (if resources available) or to suspension.

Symptom C2: PVR, PPR, PSR $<$ min. threshold (i.e. consolidation is getting exhausted).

Symptom C3: MVR < 1 (i.e. perceived value being added to the model by consolidation is less than resources being spent).

Actions (C2 and C3) :

- Conclude that feasible quality has been reached (if the values for PV, PC, PP, PS are good).

- Terminate this part of the project as hopeless, or at least backtrack to some previous decision (if the values for PV, PC, PP, PS are bad and it is impossible to see any way out).
- Switch to expansion (if the value for PC is worse than PV, PP, PS).
- Switch to other techniques or to additional language training (if one or more of the values PV, PP, PS are unacceptable; concentrate on techniques applicable for the quality aspect which is most pressing).
- Involve other participants (if one or more of the values PV, PP, PS are unacceptable).

Heuristics to guide process evolution

By aggregating collected information over several projects, one can also find heuristics to guide the evolution of the modeling-process. Important questions are:

- What specification languages, techniques, tools etc. seems to be appropriate for various categories of problems and for various stakeholders in the organization?
  Based on the user-modeling in connection with explanation generation, one can retain the information about which participants have comprehended what parts of which languages, for the use in later projects.
- What seems to be the optimal chunk size for various categories of problems using different parts of the modeling languages?
- What seems to be the optimal statement growth ratio for various kinds of problems using different parts of the modeling languages?
- What team constellations seem to be good for various kinds of problems?
- What kind of knowledge seems to be in shortage in the organization?

The results could be used to evaluate both approaches at the cycle-level and the composition of the entire process, i.e. suitable breakdown in cycles. In addition, post-project evaluations and evaluation based on actual usage of the system can be taken into concern. This is a large topic, and we will not go into details of this here.

Overview of presentation of methodology: When looking upon modeling in devtenance-projects in more detail we have divided the discussion in three parts:

- Conceptual modeling in development project with no preliminary conceptual descriptions.
- Application of the conceptual models in usage support.
- Conceptual modeling based on an existing conceptual descriptions.

The first part will include the main discussion, whereas the third part will only contain the differences when devtenance is based on using already developed conceptual models from the beginning of the projects. Apart from this, many of the same guidelines will apply to 1 and 3. The treatment will use terms from the quality framework for conceptual modeling presented in Chapter 6. In addition important aspects will be illustrated using the case study described in Appendix A.

10.2.2 Devtenance with no Preliminary Conceptual Descriptions

In this situation, one start with a specification base that is empty with regard to the application system that is of interest, and potentially a set of unsatisfied investigation reports in the COISIR base. Due to the simplified situation that we look upon here, the following will have many similarities to a traditional development project only taking one application system into account.

Preparation for modeling

Based on the project establishment the overall scope of the project has at this point been decided. On this background, one needs to identify $\mathcal{S}$, the set of stakeholders to the project. In general, stakeholders of a devtenance project can be divided into the following groups [211]:

- Those who are responsible for design, development, introduction, and maintenance of the CIS, for example, the project manager, system developers, communications experts, technical authors, training and user support staff, and their managers.
- Those with financial interest, e.g. those responsible for the application systems sale or purchase.
- Those who have an interest in its use, for example direct or indirect users and users managers.

In the conference case, the identified stakeholders were the organizing committee, the project group, the program committee, which would be direct users of some parts of the system and not only indirect as in traditional conference systems. Other potentially direct, but mostly indirect users were those receiving call for papers and call for participation, the subset of these being the contributors and/or conference attendants, the supporting conference organization (SEVU), and finally the publisher of the proceedings (Chapman & Hall). For an extended project that would include making the system into a share-ware solution, several other potential stakeholders, i.e. potentially future program and organizing committees and different conference arrangement organizations and publishers could be identified. The overall actor-model for the conference organization is presented in Figure 10.6.

Having identified the stakeholders, it is important to select $\mathcal{P}$, the participants of the project: According to Mumford [230] user-participation works best with a two-tier structure of a steering committee and a development group. This kind of organization is also often used when applying more traditional methodologies such as Method/1. The steering committee will set the overall guidelines for the development group when it comes to use of resources. It's members will typically include senior managers from affected user areas, senior management from the CIS-department and senior trade union officials. The development group will consist of representatives of all major groups of stakeholders of the project. All in the design group except system developers should be selected or elected in a way that is acceptable to their constituents and seen as democratic. One should be especially careful to avoid the situation where a group of end-users are 'represented' by a middle-level manager which himself has no close knowledge of the detailed tasks to be supported by the CIS [162]. If the project has been started as a result of a search conference, it is natural that at least parts of the group volunteering to work with this area continuous in the project group. Seen from the point of view of social construction, we recognize that the internal reality of each individual is necessarily internally held and hence reflect individual variation. It is nonetheless useful to distinguish those elements that through socialization, interaction, or negotiation individuals have in common. It is these collective cognitive elements that individuals draw on to construct and reconstruct organizational reality. Gjersvik [102] terms this a local reality on a group level. This is not the sum of the individuals local realities, nor is it shared group reality. The group's local reality is a way of acting in relationship to the organization. The way of acting is developed inter-subjectively among the individual actors of the group. Examples of groups holding a local reality can be the members of a department, the managers, or the union members. They develop this group local reality because they have common work experiences, and because they identify with each other in the organizational context. In his case-study Gjersvik identified four distinct group realities: Managers, supervisors, shop floor workers, and the

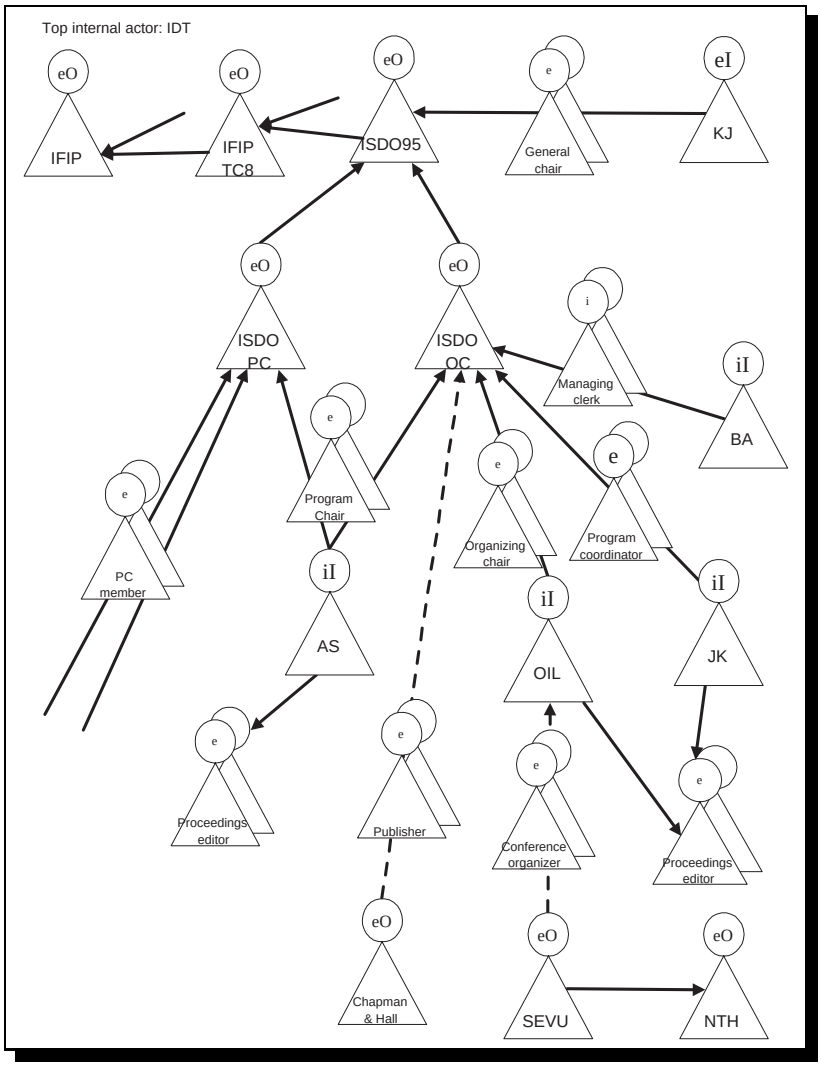

Figure 10.6: Overall actor model of ISDO95

union. Relevant dimensions in such discussions are:

- Organizational level.
- Department.
- Educational background.
- Traditional decision makers or not.
- Experience in the organization.

Mumford claim that there should be a representative from each major section and function, each grade, age group, and sex if possible, in addition to the system developers.

In addition can geographical location, experience with computers and similar applications and methodology, and attitudes to change be used as dividing lines.

In the case-study, the "steering committee" consisted of AS[1] and a student supervisor MH. The system developers will be represented as group GR, which consisted of students in the 4.year at NTH, having mostly a similar educational background. Although individual differences obviously existed, we will for simplicity treat them as a unity. In addition, JK and OIL functioned as

[1]For brevity and privacy, we only indicate the initials of individuals.

users to the system, JK being the prime direct user, at least of the first version of the system. AS also functioned as a provider of information and as an indirect users. Although not finished at the time, the PPP CASE-tool will here act as the main technical actor of the audience. The main social actors in the project and their relationships are given in Figure 10.7.

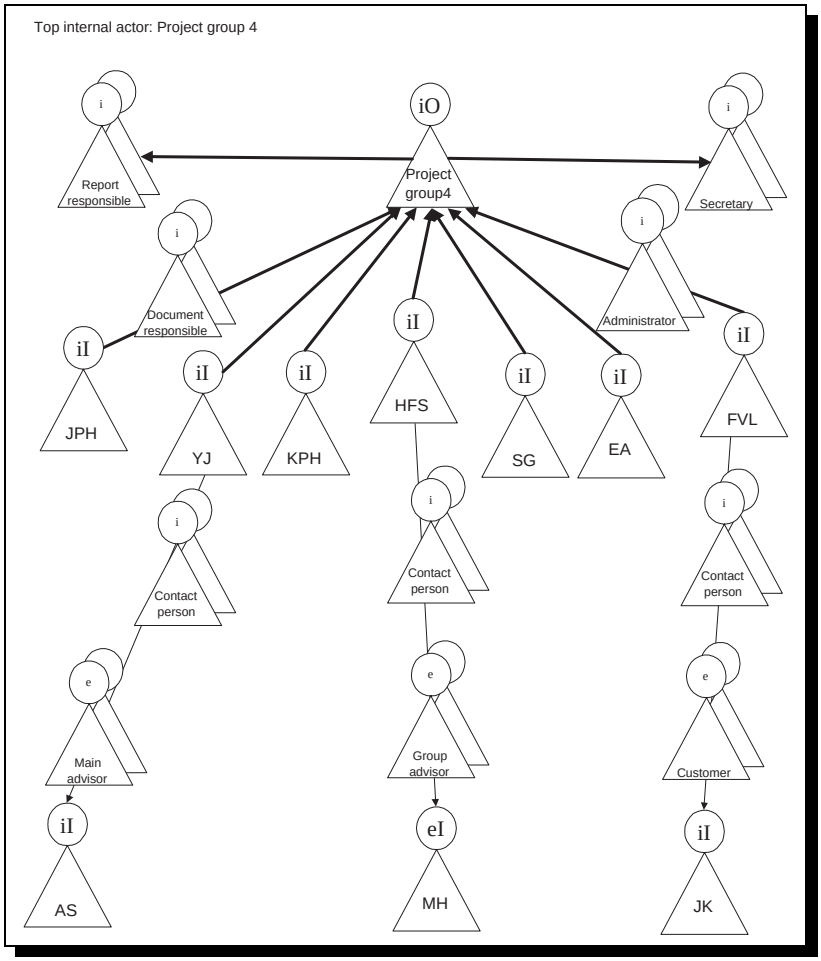

Figure 10.7: Actor model of project-participants

$\mathcal{A} = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}\}$

where A_1 = AS, A_2 = JK, A_3 = OIL, A_4 = JPH, A_5 = YJ, A_6 = KPH, A_7 = HFS, A_8 = SG, A_9 = EA, A_{10} = FVL, A_{11} = GR, A_{12} = ISDO, A_{13} = PPP.

GR = $\{$JPH,YJ,KPH,HFS,SG,EA,FVL$\}$
ISDO = $\mathcal{P}$ = $\{$JK,OIL,AS,JPH,YJ,KPH,HFS,SG,EA,FVL$\}$

Three distinct realities could be distinguished. The system developer group GR, a mature professor AS, and two young researchers connected to the research group of AS, JK and OIL which had a comparable background, although with different requirements to a system of this sort.

To simplify, we can say that at the start of the project the main involved actors had the following relevant knowledge:

- General system development knowledge $\subset \mathcal{K}_{11}$

- { General system development knowledge, PPP knowledge, Overall conference organizing knowledge } $\subset \mathcal{K}_1$
- { General system development knowledge, Specific knowledge on WWW and Ingres, PPP knowledge, Specific conference organizing knowledge } $\subset \mathcal{K}_2$
- { General system development knowledge, PPP knowledge, Specific conference organizing knowledge } $\subset \mathcal{K}_3$

Before starting on the modeling effort, a preparation period to organize and plan the project take place. This period also usually includes an initial learning course for the participants, depending on their current knowledge of the modeling languages and the domain. Whereas the system developers might use traditional ethnographic techniques such as on-site observations of ongoing activities, open-ended interviews, mappings of who is doing what and where, exploration of roles and responsibilities, identification of tasks, goals, and chains of accountability, and examinations of manuals, job descriptions, directories, division charts, and other organizationally related documents, the user-representatives will normally need training in the methodology, and its purpose, in addition to the use of the conceptual modeling languages. The training at this point should concentrate on the most important parts of the languages, so that the participants are able to model the most common situations. Learning the more advanced parts should be done on an as needed basis during the project [98]. If a user-model is to be applied in connection with explanation-generation this can be initialized at this point.

In the conference case, the project group looked upon existing material describing professional conferences, and took contact with SEVU to get more information regarding the practical arrangements of a conference, since it appeared that this kind of knowledge were lacking among the original participants. Thus, one had here an example of inter-subjectively agreed knowledge incompleteness that was remedied by contacting external experts in the field.

Another part of the preparation phase is the preparation of arenas for dialogue and modeling. This can be both the preparation of technical solutions, i.e. live-boards, meeting-room system, or other co-located or distributed, synchronous or asynchronous groupware solutions according to what is deemed needed and is available. Another possibility is to use rooms with plastic walls which one are able to write/draw on or can paste and remove paper on in order to illustrate different aspects as has been used in Tempora and in the ABC-method [333] for preliminary modeling. These models can then be transferred to a more persistent medium such as a CASE-tool to increase the physical quality of the model in the form of persistence and distributability and to enable further advanced treatment of it.

At this point, selection of which languages to use at which stages are also partly decided. In our case the total set of languages to use for conceptual modeling is more or less given, i.e. $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 \cup \mathcal{L}_4 \cup \mathcal{L}_5 \cup \mathcal{L}_6$ where L_1 = ONER, L_2 = PPM, L_2 = DRL, L_2 = Rule relationships, L_5 = the Actor modeling language, and L_6 is the set of links between model elements in the different languages. For further details regarding this division of the PPP languages, see Appendix D.3.

Below, more general guidelines for which languages and part of languages that have been found to be applicable at each stage are presented. The guidelines are partly based on practical experiences from the Tempora-project.

In the conference example, the following was decided upon:

- Development of conceptual models of the perceived current situation: Here modeling of a typical conference of this sort was performed. Languages to be used: Informal rules, rule-hierarchies, the actor-modeling language, and PPM including ports. Limited use of speech acts modeling.

- Development of conceptual models of the perceived improved situation: Modeling the same aspects as above, but with the understanding that some of the functionality should be computer supported. Applying the same modeling languages as above except speech act modeling, but develop the models in more detail. Also include ONER-modeling. Only updating the actor models if necessary.
- Development of conceptual model of the future CIS: Decide which processes should be supported by the CIS, and model these in more detail including the modeling of DRL-rules and PLD's as appropriate, in addition to further formalizing of the other models.

As will be seen below, this is somewhat different from the guidelines below, which indicate that one must apply modeling languages and techniques according to the specific situation.

Modeling of the perceived existing information system (EIS)

As a starting point for modeling, the root-definitions [51] for all participants are established using the CATWOE-technique (see Chapter 4.2.10). Starting out with the CATWOE-analysis is deemed important to enable the construction of individual models for the participants. When new persons are included in the audience, at a later stage also their root definitions should be established. The questions used in the CATWOE-analysis are similar to what Gause and Weinberg terms 'context-free questions' [98], which they also propose to ask very early in development.

The CATWOE-analysis gives starting points for several models. It defines actors and roles more clearly. Actors and roles are also found during the stakeholder identification, but additional actors and roles might be identified here. We will discuss further modeling of actors and roles primarily as part of the process and data modeling described below because of the close links between these. It can be a start of process models, indicating the main activities, and one can also use them for identifying major entities for an ONER-model. Most importantly, some of the high-level goals of the different participants which can be used initially in the goal-hierarchy are established through the CATWOE-analysis.

As an example, on answering the question:

> **Who** is doing **what** for **whom**, and to whom are they **answerable**, what **assumptions** are being made, and in what **environment** is this happening?

OIL gave the following response:

- Customer (the whom): The organizing chair of the conference and SEVU
- Actor (the who): GR and JK.
- Transformation (the what): Good communication and coordination between SEVU and the organizing chair.
- Weltanschauung (the assumptions): A well-organized conference is of major importance to the research group.
- Owner (the answerable): AS.
- Environment: The university.

As part of the CATWOE-analysis, the following goals were identified. The sources of the goals are indicated in *italics*. The numbering is similar to the numbering used in Appendix A.1.2.

- **R1002**: "It is recommended for the group to get a good grade on the project" *GR*.
- **R1003**: "It is recommended for the group to make the customer satisfied" *GR*.
- **R1004**: "It is obligatory for the group to keep within the time-budget" *GR*.
- **R5**: "It is recommended for the organizing committee to create interest for the conference" *JK*.

- **R152**: "It is obligatory for the program coordinator to keep track of papers and reviews of papers" *JK*.
- **R902**: "It is obligatory for the organizing chair to have good cooperation and coordination with SEVU" *OIL*.
- **R4**: "It is obligatory for the organizing committee to create interest for the conference" *AS*.
- **R504**: "It is recommended for the conference system to support other conferences than ISDO95" *AS*.

This initial situation is depicted in Figure 10.8. Rules and arguments are given in boxes, using arrows for relationships annotating these with **O**, **R**, **P**, **D**, **F**, and **REL** (for an undetermined relation). The sources of both rules and relationships are indicated in ellipses. We are not using the actor symbol, since this source-relationship is really on a meta-level of modeling, and not on the same level.

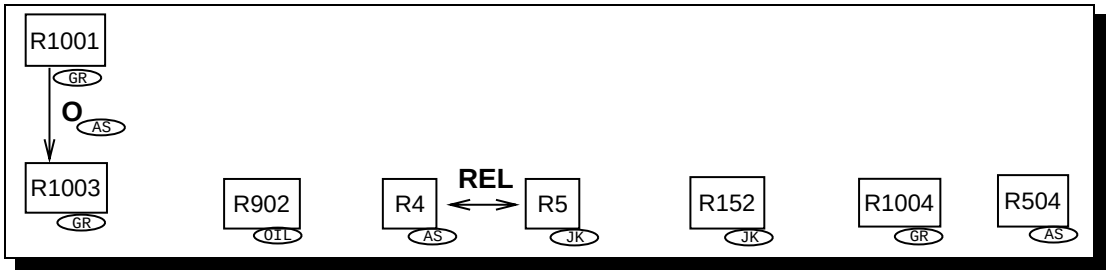

Figure 10.8: Rules based on CATWOE analysis

Goals and rules that appear at this stage are not necessarily on a highest level, thus one can work upwards from the existing goals. Some rules identified on this basis which were generally applicable for the first modeling task were:

- **R3**: "It is recommended for a research group to arrange conferences". This goal recommends the goal **R504** according to *AS*, with the additional argument **A1**: "Many conferences of this kind are arranged. If we can come up with a general system that can be used in other conferences arranged in the community, this will be a contribution from us to the community".
- **R501**: "It is recommended for our research group to arrange this particular conference" *AS*. This is motivated both by rule **R3** and other goals not given here.
- **R401**: "It is recommended to apply the services of a local conference organization" *OIL*.

We will indicate how the hierarchy can be developed downwards later in the chapter.

For the further modeling, the development group is first divided into homogeneous groups for the first modeling efforts similarly to how the division is done in search conferences. This is contrary to what is done in e.g. ABC and JAD [12] where the use of inhomogeneous groups is proposed from the start. The main argument for homogeneous groups which in one extreme include one user-participant and in the other extreme all the participants, is to allow time for the individual to develop his own model to address problems of model monopoly. In the case-study of Gjersvik [102], the local reality of management was more easily transferred into an application system because of their training in thinking in abstractions due to their generally higher education. Problems of this kind with JAD have also been reported in [45].

In any case each conceptual model "fuses" at least two internal realities. If the users had been excluded from the analysis process, the developer would have had to bridge both his own and the

reality of someone knowledgeable in the domain. The situation is symmetrical for users. If they undertake the development themselves, they must cope with the presumptions that is embedded in the modeling languages and tools. If users develop also their own languages in every case, reuse will be minimal, and it will be difficult to support model and application system quality. We suggest to have a user and a developer to be the minimum unit of cooperation, where the developer will be supposed to be able to use a larger part of the conceptual modeling languages to be able to apply techniques to improve semantic and pragmatic quality of the models. We will below assume that more than one model is created based on the local reality of parts of the audience, thus that there will be a need to merge different models , a process which is supposed to include both negotiation and learning.

In the case study, models where first developed individually for actors, thus developing $\mathcal{M}(EIS)_1$ as an externalization of $\mathcal{K}_1$, $\mathcal{M}(EIS)_2$ as an externalization of $\mathcal{K}_2$, and $\mathcal{M}(EIS)_3$ as an externalization of $\mathcal{K}_3$

The model-integration strategy was originally to first integrate $\mathcal{M}(EIS)_1$ and $\mathcal{M}(EIS)_2$ into $\mathcal{M}(EIS)_{1,2}$, and then integrate $\mathcal{M}(EIS)_3$ with $\mathcal{M}(EIS)_{1,2}$ into $\mathcal{M}(EIS)_{12}$.

To assure organizational learning $\mathcal{M}(EIS)_{12} \subseteq \mathcal{M}(EIS)^1, \mathcal{M}(EIS)_{12} \subseteq \mathcal{M}(EIS)^2$, i.e. whereas JK and AS had interest in the complete model, OIL had only interest in the part that was an externalization of his knowledge regarding the practical organization of the conference.

Although education of the participants has been performed on the use of parts of the languages that has been selected for use at this stage, different persons should use different languages according to their ability to deal with abstractions. Experience indicate that it might often be easier for workers to tell their stories using the parts of the process model which are able to deal with the instance level. These instance-level stories can then be abstracted up to class-level ones. A data-model on the other hand represents only class-abstractions. For others, e.g. management, which traditionally will be better suited to deal with abstractions, a data modeling language can be more suitable for constructing models [102]. By using driving questions though, and by applying the links between models actively, it should be possible for all participants to build up models in the different languages that are used and integrate them.

As an example of the modeling on the instance level, Figure 10.9 indicate the situation regarding the distribution of CFP's seen from the point of view of the actor JK.

This could have been based on the following story.

> "The other day, I got a request from NN to send him the CFP for the conference. The CFP had I earlier got from AS who originally made it. After updating the address-list over people that wanted to receive information about the conference, the CFP was sent to NN"

This model can then be generalized as earlier depicted in Figure 9.3.

The modeling on this stage is divided into five main tasks. That does not mean that the tasks are done sequentially, one typically will go back and forth between different tasks. We will first describe the general guidelines, based on both the case studies and also on guidelines developed for Tempora, and then indicate briefly how modeling was done in the conference case. The tasks are:

- Goal modeling.
- Data modeling.
- Process modeling including actor/role modeling.
- Rule modeling.

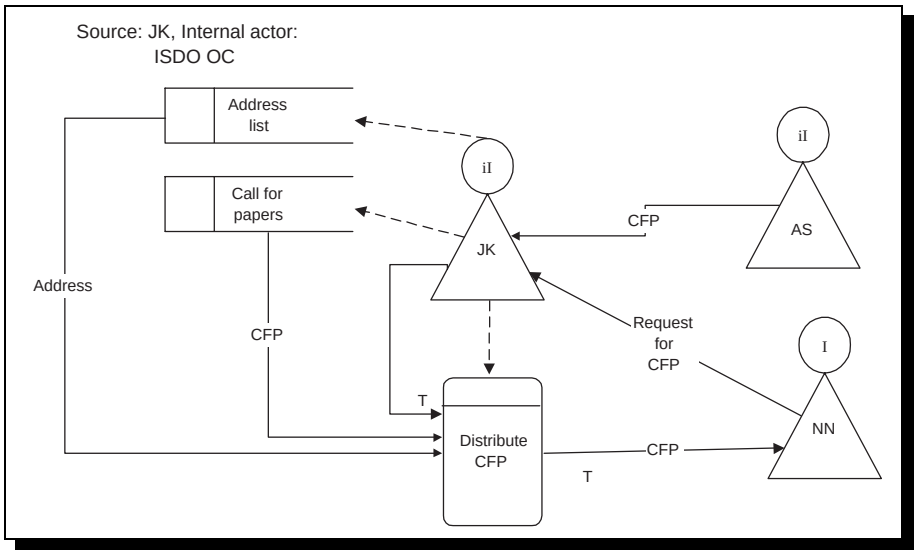

Figure 10.9: PPM describing CFP-distribution at the actor-level

- Using techniques to help increasing the pragmatic, perceived semantic, and social quality of the models.

Goal modeling: Objectives as perceived by the participant are described in connection with higher level objectives e.g. overall strategies. The start of the goal-model is based on the results from the CATWOE-analysis as illustrated above. As goals and rules are identified (see rule-modeling below), one try to incorporate them in the goal-model using the deontic relationships.

Both for the individual models and the integrated model, rule and goal-modeling was used in the conference case.

PPM modeling: PPM modeling by non-technical participant should be performed without applying ports and not from the start distinguishing between triggering and non-triggering flows. Thus the PPM as used here resembles regular DFD. The modeling of process logic should not be considered. On the other hand, if it is necessary for the comprehension of the model, the developer might on parts of these models use all of PPM including PLD or rules to enable execution, code-generation, and explanation. When high pragmatic quality has been achieved, one will continue modeling temporarily discarding the extra details added using an appropriate filter.

Guidelines:

- The process modeling do not necessarily start from a top-level process, i.e. it is more convenient for certain groups to deal with the detailed processing in one area.
- A process corresponds to a business process, not an automated process of the target CIS. An actor or role can be used to indicate in more detail who supports the process.
- The scenario to the ONER-model if any conveys an abstraction facility for all kinds of stored data. One know that the data is stored in some form but a further description is postponed. The contents of ONER-scenarios may be specified by simple access expressions or even left out.
- Flows at a high level of decomposition are not detailed until the process model at lower levels of decomposition are well understood.

- In the beginning, no distinction is made between control flow and data flow. Then one concentrates on the identification of control flows.
- Filtering is useful to support the construction of port structures. This is particularly true when the number of flows entering or leaving a port is high.
- Each process must at least be associated with one event-action rule, although this can be described informally.
- Flows can be specified with regard to an ONER-model, but do not need to.
- Actors and roles in the diagrams can be either external or internal to the organization. One do not apply the traditional notion of 'external entity' at this point.
- If one are indicating speech-acts, one should try to complete the conversations, and structure the process models around these.

In the conference case, one initially developed process-models in the DFD-style, with the addition of indicating supporting actors. Since the participants knew PPM-modeling well, also triggering and terminating flows and ports were indicated. Models where made on smaller areas, trying to synthesize an overall process-model through model-integration. No ONER-model was made at this point, thus stores and flows were not described in much detail. Limited speech act-modeling was done separately.

ONER modeling: Concentrating on the modeling of main entity-classes and relationship-classes, including attributes and values if necessary. Cardinality constraints are specified if it is deemed necessary. Similarly to above, the system developer might include additional model-details to be able to use the techniques for improving the model-quality.

Some guidelines at this stage are:

- Strict classification of entities into specialization hierarchies should not be enforced early in the modeling process. Therefore, one should postpone modeling of specialization of entity classes if they do not have any particular relationship classes to any of the other central components of the model, or they appear naturally in the discussion of roles in the organization. Alternatively, they are hidden or developed in separate views.
- Different views of the same phenomena are allowed, since different properties may be of interest in different situations.
- Redundancy and division into submodels are often necessary.

DRL modeling: Most rules are to be expressed in natural language, using a general *when-if-then-deontic-for-role/actor-consequence-else-consequence* structure using informal text in the fields.

Guidelines:

- Both informal and formal rules should be explicitly represented.
- If the number of rules specifying exceptions to a rule is high, the specialization criterion of the entity classes involved should be reviewed.
- Predicates are useful to split a composite rule into simpler rules.
- Ports together with abstraction mechanisms are useful to support the rule formulation when the number of parts making up a rule is high.
- The I/O-matrices can be useful to support rule formulation when there is a high number of combinations to be taken into account.

In the conference case, rules were only informally described at this point.

Increase model quality: In the expansion-phase of modeling, the driving-questions technique can be performed, often not even using a modeling tool in the first expansion/consolidation-cycles. When improving the physical quality of the model by transferring it to a modeling tool, only syntactic invalidity is addressed immediately by the tool.

Before switching from expansion to consolidation, checks for syntactic incompleteness can be performed, before concentrating on comprehension. In the conference case, comprehension techniques at this stage were inspection and filtering. This was judged as sufficient because of the comprehensive knowledge of the modeling languages held by the participants. This would not normally be the case and one could here be assisted by the explanation-generation facilities to become familiar with the modeling languages. One could also at this stage have added more details to the model, enabling execution and explanation generation. Based on the knowledge of what needs to be added to different kinds of models to use the different techniques, one could get an indication of the "incompleteness" of the model with regard to execution or explanation generation based on the area selected to be executed e.g. what is missing for the model to have necessary formality as discussed in Chapter 6. Based on earlier experience one could then get an indication of the resources needed for preparing for a prototype or explanations.

After comprehending the different models, perceived validity and completeness can be looked into. The driving questions can be used also here, but in a way to get an indication of that there are incompleteness, rather than to start on a new expansion-phase immediately. One should also apply techniques for consistency-checking of the models here to support this process. If the perceived semantic quality is regarded as satisfactory on this level, one can return to a reconciliation of the models written in the different languages to check such things as consistent naming before trying to integrate models based on the views of different individuals. Throughout modeling one should also build up a thesaurus of terms which are negotiated in parallel to the development of models [27]. As an example of a process model made in this way, Figure 10.10 shows one of the PPM models made based $\mathcal{K}_2$.

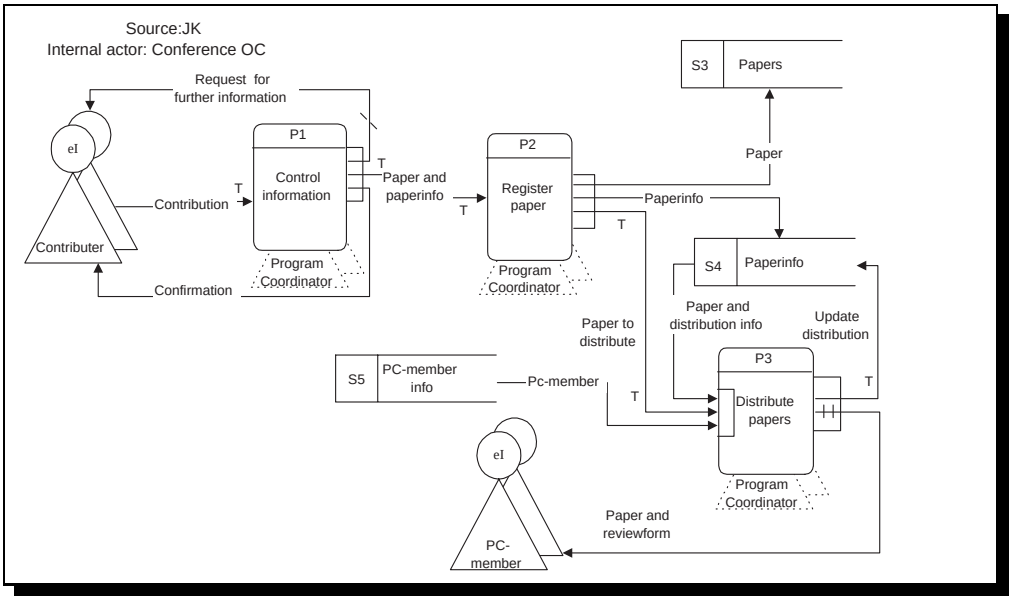

Figure 10.10: PPM describing reception and distribution of papers based on $\mathcal{K}_2$

When integrating models made by different actors, this is one area where model integration

techniques can be useful. Totally automatic methods are not applicable in this case, merging will necessarily involve human judgement and negotiation. When comparing models which have no common predecessor made by persons from different parts of the organization, it is more important to find overlap than differences between models. Before doing negotiations to get a common model, the comprehension of the models made based on the internal reality of other participants must be assisted. If all models are made in languages that all involved participants are familiar with, one can start by having the participants trying to explain the models made by others. In this case it might be more cost-efficient than above to add details to parts of the models to support comprehension techniques such as execution and explanation generation.

If the models are developed using parts of the languages that the other participants have not yet used themselves, language training in this area is given in the preparation phase. Initially the other models are regarded as fully uncomprehended, but totally valid, i.e. $\forall i, j \; i \neq j, \mathcal{M}_j \setminus \mathcal{I}_i = \mathcal{M}_j$. Comprehension is then incrementally achieved and validity is questioned in the process. In this way the participants can learn about other parts of the organization at the same time as they attempt to externalize their local reality. When comprehension is achieved, model integration of different models are attempted. This process might result in the discovery of conflicts as indicated in Chapter 6. Of special interest is the discovery of inconsistencies, which might need to be reconciled. We will return with an example of this later in the chapter.

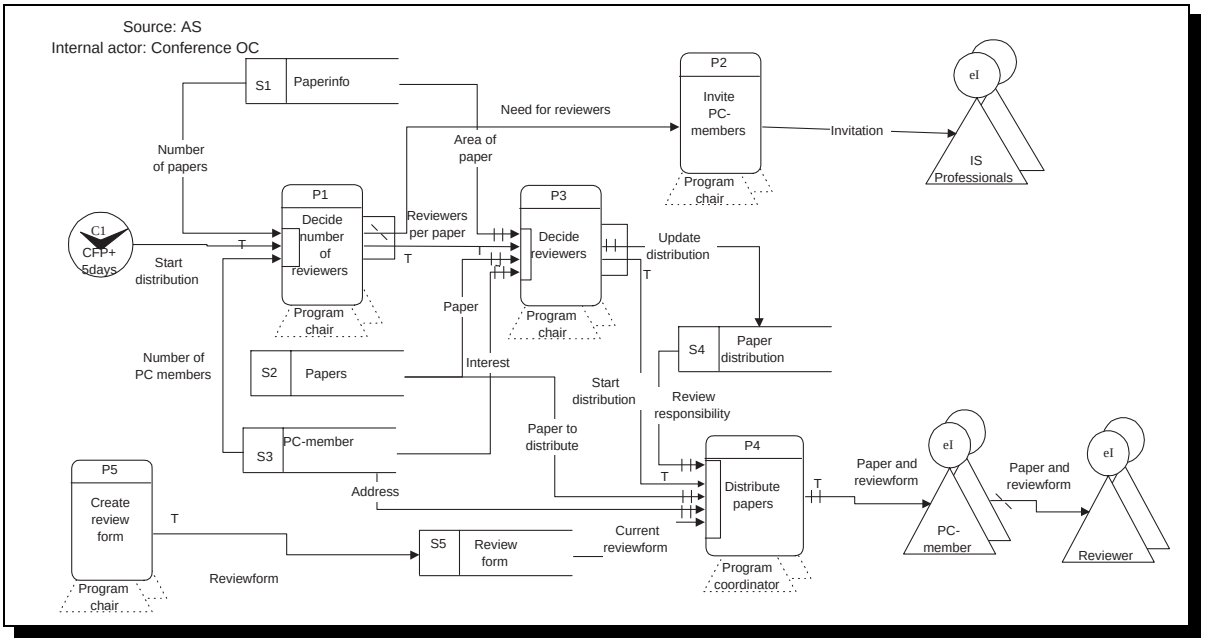

Figure 10.11: PPM describing distribution of papers based on $\mathcal{K}_1$

An example from the case-study is the comparison of Figure 10.11 where the source was AS with Figure 10.10. We can recognize one common process (Distribute papers) and one common role which receive papers (PC-member), whereas the additional processes in the diagrams regard different things. Also some common data-stores can be recognized although using similar, but different names. How this process interact with other processes differs between the models, and need to be reconciliated. The issue is if one should distribute papers to reviewers on a paper to paper basis as they are received from contributers as suggested by the model of JK, or as one process which is first run after that all papers have been received, as suggested by the model of

AS.

After integrating the models, one should look upon the amount of the resulting statements in the combined model that was the source of each of the participants, comparing this with the number of statements in each of the original models. If only a small percentage of the statements in the original models of one or more of the participants are retained in the official part of the joint model, this might indicate a situation of model-monopoly.

After model-integration, one might go back to a phase of expansion based on the new combined actor, which are followed by a new consolidation phase. Alternatively, one might like to reconcile this model with another model right away.

These kinds of processes will continue until there is a common model of the participants perception of the current situation. In a sense this has created a new organizational reality for the development group, although it do not at this point apply generally in the organization. This do not mean that all differences in the models are removed. It is neither to be expected or wanted that even after a mutual learning process, the participants look upon the organization in an identical way. The overall assumptions that surfaced in the CATWOE-analysis have not necessarily changed. The representation of this variety is most easily done in the goal-model with its possibilities of explicit representation of inconsistencies between views. Since the goal-model also links to and motivates statements in other models, it will be the main arena for negotiations.

In the case study, models based on the knowledge of JK, AS, and OIL were created. Then, the models made based on the knowledge of JK and AS were reconciled. At this point it was apparent that one needed to concentrate on the paper-process only, and thus a further reconciliation of this combined model with the model based on $\mathcal{K}_3$ was not performed, although it was planned. A reconciled overall process model of the paper-process is given in Figure 10.12.

Modeling of the perception of a future information system (FIS)

The main task in this phase is to establish the requirements for a new or expanded information system.

When starting out on this modeling effort, one has as an outset a model of the current situation as it is perceived by the participants, and potentially also several COISIRS regarding this area in the COISIR-base. In addition one must take into account how the perceived future environment of the organization will look like to not only change reactively, but also pro-actively.

The guidelines for the use of modeling languages in this phase has much in common with the guidelines that were presented in the previous phase when it comes to the models that are produced and which parts of the modeling languages that are normally used.

Also, the general methodology, with homogeneous groups developing models of the perceived improved system before merging these are kept in principle, although the need for many different separately developed models probably is less, since the participants have been able to develop models based on their local reality, models which are at least partly kept. A difference is that the separately developed models in this phase will have a common predecessor, which should make the situation for merging easier. One will also probably use the techniques for improving the pragmatic quality of the models to a larger extent in this phase, to be able to run what-if analysis of different scenarios, since the cost of doing this in areas where one might later on perform further CIS-modeling is less from an overall project point of view because of the potential reuse in the next phase.

In the conference case, due to the decision of not including support for practical arrangement of conferences, the audience was reduced to only consist of two users, AS and JK at this stage.

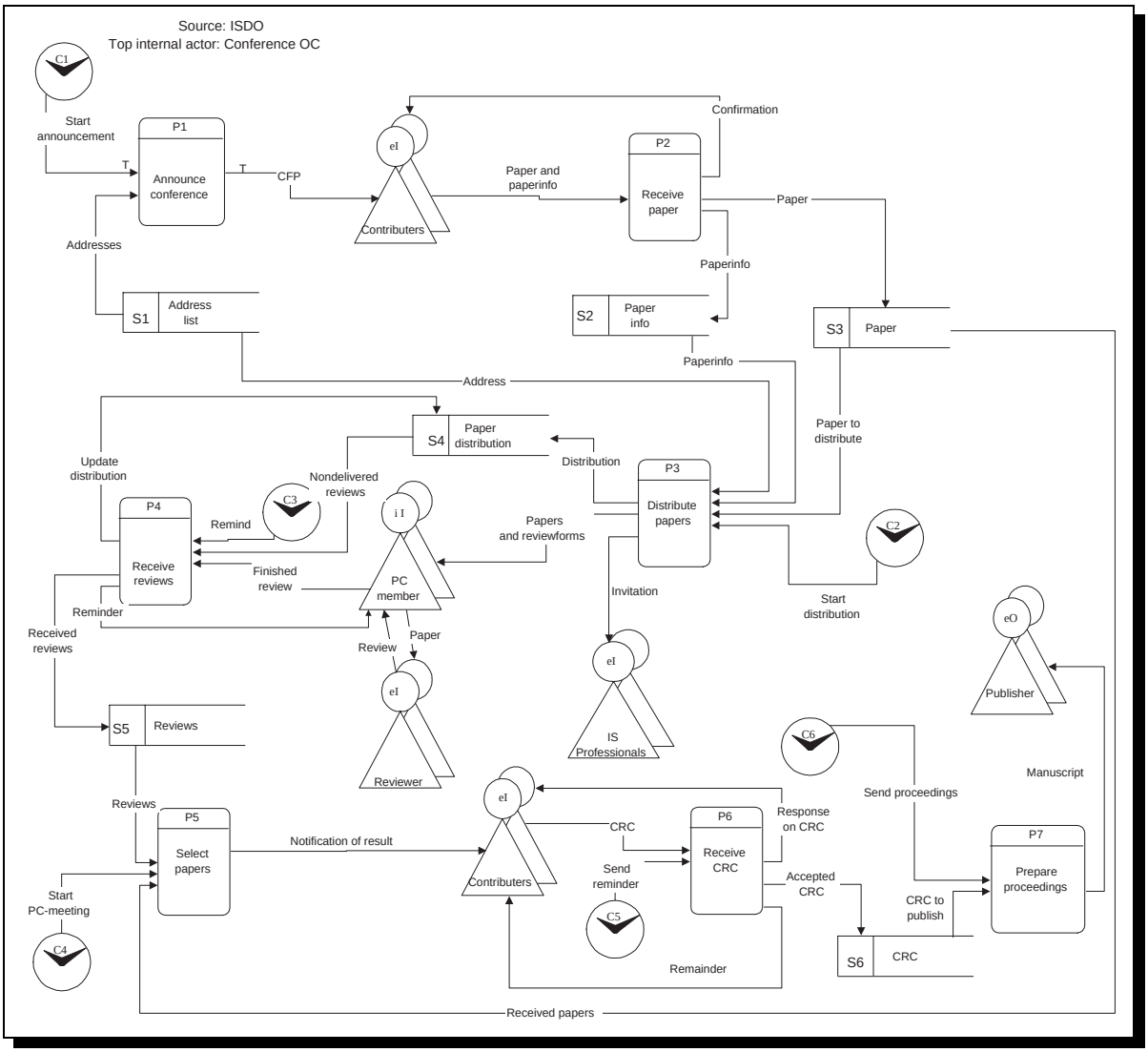

Figure 10.12: PPM describing the paper handling

Whereas rules were still made having indicating the individual source, the PPM-models and the ONER-model which were developed in this phase were developed jointly. The PPMs are presented visually on the actor-level, to make the models more concretely applicable to the current situation. It was also modeled in which roles the actors acted to have the models more generally reusable.

The domain is here changed from the previous phase, since one is now modeling a future non-existent reality, and not the existing as it is perceived. Although one should not be too constrained by the existing situation when developing new solutions, one would usually like to retain certain of the existing rules for the situation. Many of the actors will also be the same, whereas one should be careful with adapting existing process diagrams uncritically. Thus based on the limited scope of modeling, one should decide upon the parts of the models which should be retained in a new solution as a first step. This model then contains a part of the new domain, and it is possible to specify completeness and validity of the new set of models with respect to these models, by having traceability between models. This “baseline” can be changed as further modeling gives

new insights, and is only a part of the domain, thus completeness and validity can not be judged on this basis only. Other metrics for completeness and validity of rule-hierarchies have already been discussed in Chapter 9 and will not be repeated here.

In the conference-case, one additional area that was not discussed in the modeling of the perceived current situation, the general handling of an address-register, was included, see Figure 10.13

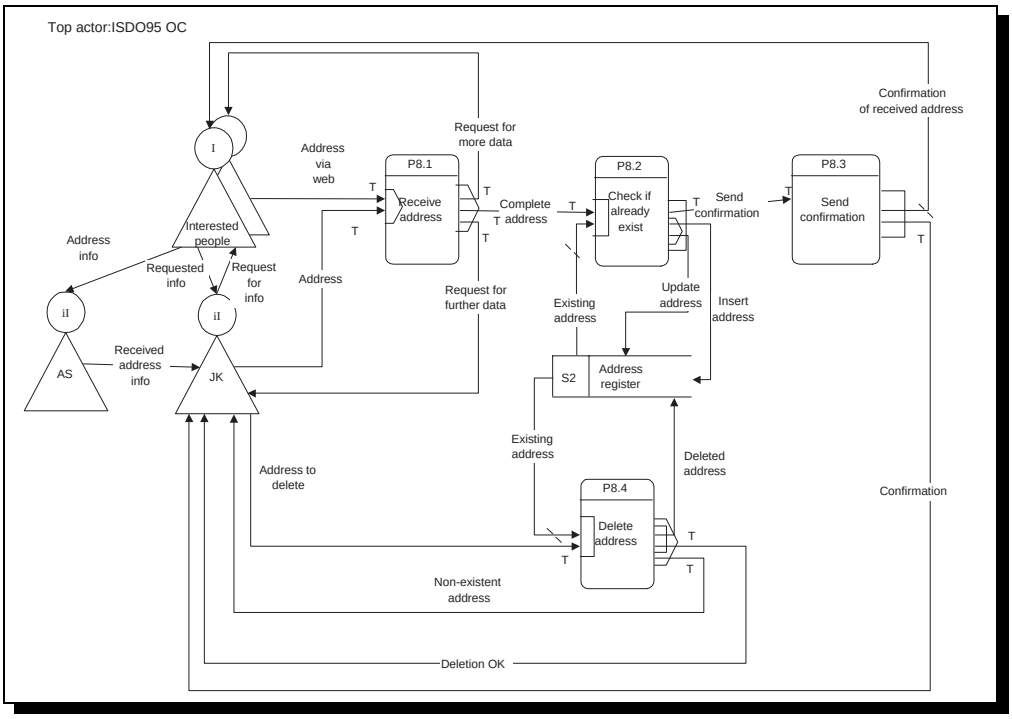

Figure 10.13: PPM describing address-registration

In other cases, part of the new model grew out of the models prepared in the previous phase, see e.g. Figure 10.14 which contains a further specification of process P4 in Figure A.24 given in Appendix A.

Modeling of the future CIS-support (FCIS)

The main tasks of this phase are:

- Determine the automation boundary, and the part of the application system that should be developed in this project if one are using an incremental development approach.
- Model and design the target application system.

In the conference case, one opted for incremental development, thus only some of the application system was to be developed in the first project. Whereas e.g. all the processes in Figure 10.13 were to be supported by the new application system C4, only process P5.2.1 of the processes in Figure 10.14 was to be supported as part of the first project.

Based on the decision on automation boundaries, one has a similar situation that open for traceability between models as discussed above. In practical cases, the model decided upon in the previous phase with the indication of the parts that are to be implemented in this project might act as a contract for the rest of the project if it is possible to decide on this, and thus should be adhered to in a larger degree. This does *not* mean that these models cover the entire domain, additional

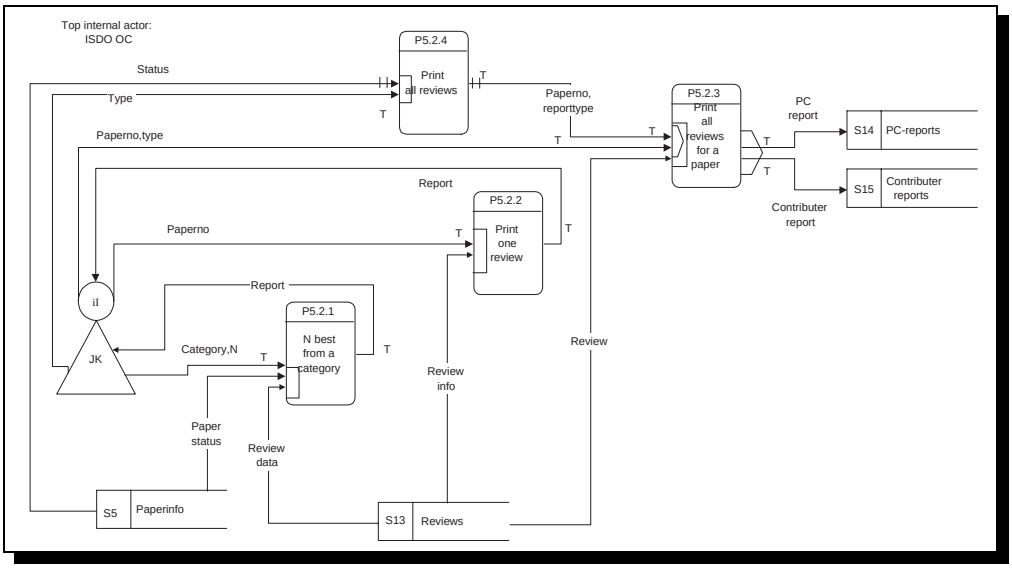

Figure 10.14: PPM describing specifications of reports

factors according to the changed nature of modeling (i.e. the modeling of a future CIS-solution) will appear such as additional non-functional requirements to the future application system.

FCIS modeling involves the following, not necessarily sequential activities:

- Determine boundary of future CIS. This can be done using the PPM model from the previous phase as basis from which the parts to be supported by the CIS are selected. The new applicative actor is indicated to support these processes. It is also indicated which actors support the other processes. Also for stores and flows, support relationships can be specified. Based on this selection, relevant parts of the developed ONER and DRL models can be derived. In selecting the areas for implementation, the priority information indicated in the deontic operators in and between rules can also be useful.
- Develop ONER-diagram of CIS: Based on the relevant parts of the ONER, the following details are added if not already included.
  - Type definitions for all types.
  - Each type is attached to at least one entity class.
  - Further refinement of relationships.
  - Specification of all cardinality constraints.
  - Natural language descriptions of all ERT classes.
  - Specialized entity classes are defined together with the subset-relationships.
  - Specification of derivation rules.
  - Specification of methods, and description of the methods as rules or as PLD's.
- Develop PPM of CIS: The process specifications are decomposed to the lowest level of decomposition to include the necessary details of the CIS. An indication for if this is reached can be formed from the number of rules needed to describe the process. If one need more than approximately five rules to described the processing of the process, this might indicate that the process should be further decomposed. When describing the process with a PLD it is not as easy to give strict guidelines, but one might use e.g. the McCabe cyclomatic complexity to get an indication of this, since a high number here might indicate that the process contains a large set of implicit rules. Detailed discussions on code-metrics are regarded as

beyond the scope of the thesis.
The whole vocabulary of PPM is used at this stage:
Guidelines:
  - Parts of the domain not part of the target CIS, that interact with the system is depicted by the actor or role controlling the relevant processing, being either external or internal.
  - Only the parts that are to appear in the database should be described in the detailed ONER-scenarios.
  - Details of flows may not become clear until the process logic is well understood.
- Develop DRL of CIS: The whole DRL-vocabulary may be used during CIS-modeling. However, the rules should be declarative in the sense of having a logical semantics, but not an operational semantics. Links from the goal model to the other models should be completed. When needed for execution, rules are either given executional semantics, or their executional semantics are specified using PLD. One should be aware of not applying PLD's too early though, since they easily can come to cover several rules in the domain [168]. The following tasks need to be carried out;
  - Decompose complex rules into several simpler rules.
  - Formalize natural language rules in DRL. When doing this, one has noted that it is often necessary to change the rules somewhat to be able to formalize them. When using the formalized rule in the CIS one needs to reaffirm that the existing rule-relations are still valid.
  - Specify all rules that refer to the ONER model. Constraint rules will fall under one of the following categories: Population, value, uniqueness, disjoint relationship, and subset relationship.
- User interface design. User interface descriptions are developed at this stage, linking flows in the process model that crosses the automation boundary to the static and behavioral part of the UID as appropriate. The user-interface can be developed partly independently from the conceptual models, but consistency checking between the models should be possible to improve semantic quality.
- Review of the model: The CIS-model should be gone through systematically and carefully inspected by the developers and other participants. A couple of things to note here is that changes done to transfer the specification from the problem-oriented specification in the previous phase to the more product-oriented specification which are developed in this phase, one will often need to rearrange the models. This rather traditional analysis/design gap should not comes as a surprise, also when going from e.g. OOA to OOD such a gap is often experienced even when one are using the same languages [61]. This formalization also include the addition of details to the specifications that earlier have not been stated explicit, so going back to assure the perceived semantic quality is important. Other factors, such as non-functional requirements which are first of interest when discussing a CIS might also change the old specification. Finally, it is possible to create preliminary prototypes at this stage of the whole system as it will appear for the users, including the user-interface.
  Thus one can at this step perceive the use of a combination of several techniques for the comprehension of the models and the effects of the application system to be the result of the models. User-interface prototyping can be performed individually based on the UID. If events in the user-interface is also linked to inputs to processes that are described in detail, one can start with entering data in the user interface, and initiating the required event (i.e. pushing a button). One can then follow the flow of this data through the PPM using the

prototyping facilities, until the result is returned to the user-interface. In case the steps in the processing are not comprehended, it is possible to look beyond the level of processes when using the general execution mechanism. It is also possible to query the model for its dynamic behavior, using the explanation generation facilities. The filtering techniques can also be applied by only showing the processes on the path starting with a given process, and e.g. not showing ports.

It is also possible to show these prototypes to other stakeholders, and not only the participants in the project. On the other hand, if an incremental approach is followed, the application system is meant to be committed to the organization soon after all, giving the users the possibilities of having real experiences with the system, which are judged to be more efficient in coming up with constructive suggestions to improvements which then can be included in the next version of the application system.

In this phase, it is opened up for compositional reuse, being done either bottom-up or top-down. Existing reusable processes that address simple functions in the bottom of the process model can be identified and reused. Application framework based reuse is also perceivable, where the functionality of existing frameworks are compared with the agreed models of the application system within the automation boundary. This will be most useful in cases where the framework supports automatic generation of the application system for the chosen supportive technical actors, and the supporting CASE-tool do not.

As mentioned earlier, one important technique is IBIS this can be used in connection with the issues discovered by the modeling of goal-hierarchies. As discussed in Chapter 9 certain situations in the goal-hierarchy identifies the existence of an issue, which should be debated and resolved at some point if the area is to be computerized. The relevant rules can then be transferred to a separate tool for the performance of such argumentation, and the results of this can be transferred back to the goal model, potentially adding new arguments and rules to this.

Returning to the overall goal-model that we started with as depicted in Figure 10.8 this can also be developed downwards. Below, we list some of the rules being pursued regarding the choice of the target platform for the CIS:

- **R505:** ”It is obligatory that the conference system run on Unix” *JK*. This is recommended by **R152** and supported by the argument **A2**: “I (JK) will be the main user of the system, and I use Unix for other work in connection to the conference”.
- **R506**: “It is recommended that the system run on PC using MS-Windows” *AS*. **R504** using argument **A3** “Many people use PC’s” is regarded by *AS* to obligate **R506**. At the same time, this rule is discouraged by the combined effect of **R1004** and **R505** according to JK.

A goal-hierarchy only including the rules mentioned specifically in this chapter is depicted in Figure 10.15.

Here an *issue* is identified: The platform for the conference system. This issue, with the identified *positions* and *arguments* for these can be transferred into an IBIS-like tool [56], where a concentrated argumentation process on this issue can be performed. Figure 10.16 gives an overview of the situation after additional positions and arguments have been added, some of which can be transferred back into the goal-hierarchy. The *decision* of the argumentation was to choose Unix for the first version of the system, but perform a general design of the system to enable an extension in a later version of the system. Since arguments in this situation can be further motivated by additional rules, one are also able to model the assumptions of arguments as suggested in [266].

Although discrepancies are resolved, one do not totally discard the other alternatives. The pruned goal-hierarchy is showed in Figure 10.17, also indicating that **R505** is linked to an issue.

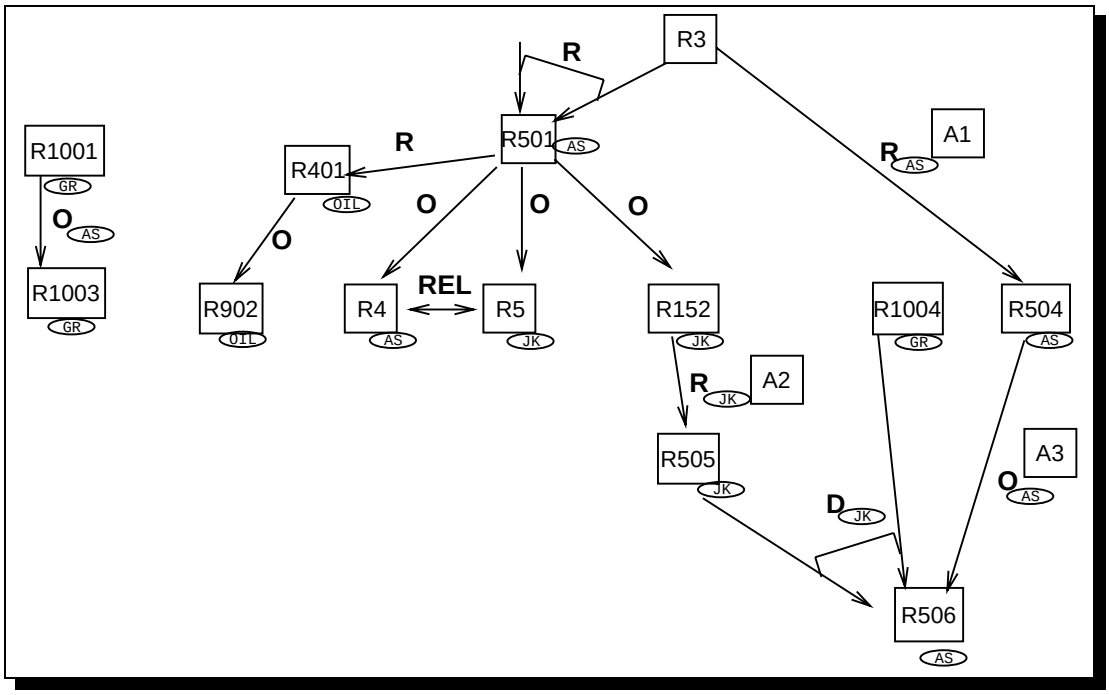

Figure 10.15: Goal-hierarchy extended from CATWOE analysis

In this way one may prevent premature closure by capturing the variety. If it appears later in the project that new factors e.g. technical constraints indicates that one should re-assess an issue, this can easily be done. If many different positions are present on an issue, this might indicate an area where changes are more likely to be done in the future. This might indicate that more effort should be used up front to prepare this area for further development at a later stage. It is also important to note the design decisions that are made based on the particular supporting software and hardware actors in the project, since it is likely that one might want to change this underlying platform at a future time. Generally, the argumentation structure which is used is believed to be a good way to capture design decision, thus in a maintenance project enable a better understanding of why the existing application system was made in the way it was.

Translation of conceptual models into an application system (CIS)

The purpose of this phase is to implement the application system. The specification of the CIS should be sufficiently detailed to enable a translation to the set of supporting actors of the application system. The translation should ideally be automated in a large degree in the case this is supported. Currently, this is only supported in a limited degree towards the run-time system of Tempora for our conceptual framework.

Transaction boundaries must be identified. The transactions are further detailed and optimization is addressed. The detailed design decisions being taken at this stage is also kept to be able to replay these if necessary in a later regeneration of the system. TEQUEL rules can be generated from the rule model, and the database schema can be generated from the ONER model and constraints expressed in DRL. The executable rules are fed into the rule manager which controls the execution of the rules.

In the future, additional execution architectures should be supported. When automatic code-

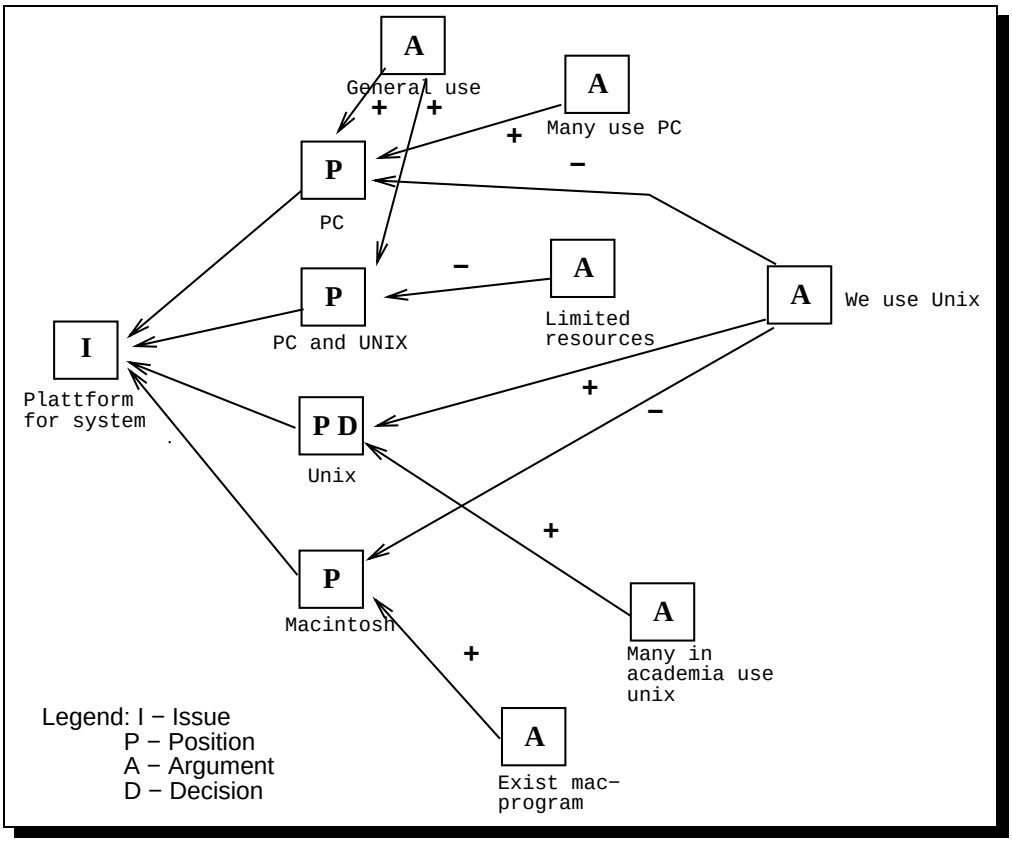

Figure 10.16: IBIS-network on the issue of system platform

generation is not supported, the application system must be partly manually coded based on the detailed specifications and design of the CIS in the previous phase.

In the conference case, the application system was implemented using C and MOTIF on top of the Ingres DBMS and WWW. Also some Unix-shell scripts were developed. Figure 10.18 gives an actor-model including the main parts of the new system.

Summary on modeling approach

Figure 10.19 can be used to summarize the approach, giving a generic overview of the conceptual modeling within a devtenance project. Each circle indicate a modeling effort based on the knowledge of an individual or organizational actor. As indicated in the top left of the figure each modeling effort can go through the modeling of several revisions and variants of the model, and the effort follows the SPEC-cycle, ending this modeling effort at suspension. This means that any modeling cycle in the same or previous major phase may be restarted e.g. in case one use an iterative development strategy. It is also indicated that a modeling effort can use a limited set of the overall conceptual framework, and use this indifferent depth [140].

When integrating two (or several), models, their own modeling efforts are in the suspension state, and the modeling of the joint model moves to preparation, setting up for merging and further modeling for A joint organizational actor, and possibly additional language training. When whole or more usually parts of the existing models are introduced, this is the first expansion activity, which immediately is followed by a consolidation phase, where one try to achieve pragmatic,

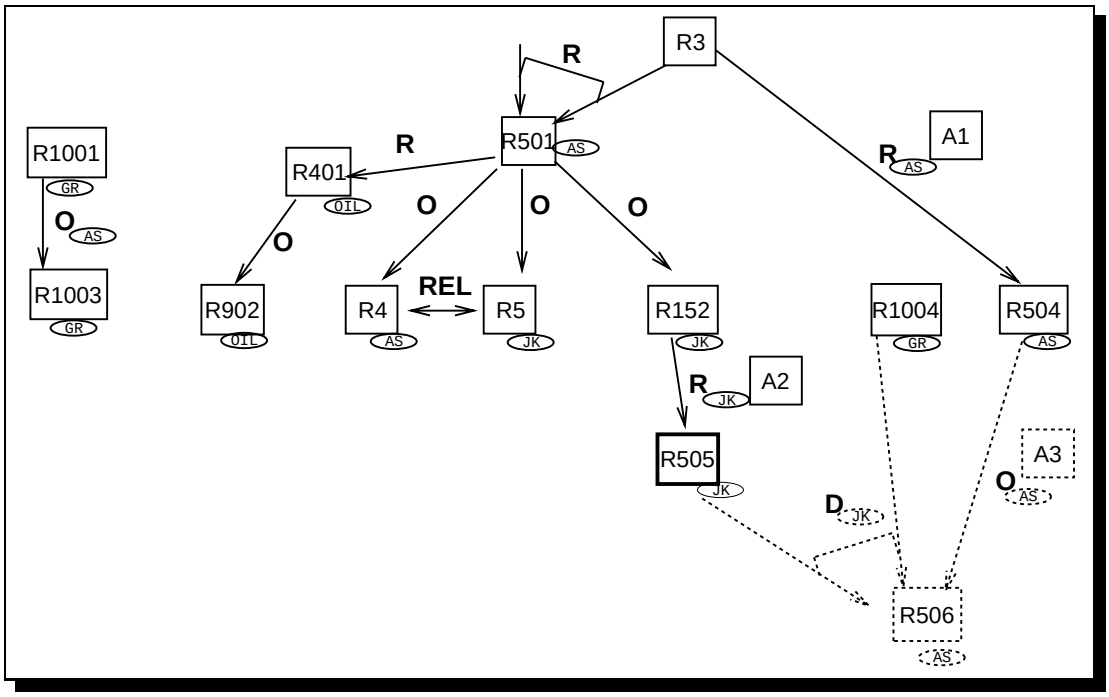

Figure 10.17: Pruned goal-hierarchy after argumentation process

perceived semantic, and social quality. One can then return to further expansion/consolidation cycles on this model. In this model, one will typically retain statements from all the predecessor-models, and also prune statements from all the predecessor models (indicated with black in the figure). In addition will often new statements be added through this modeling effort.

When the modeling of the perceived existing information system is suspended, one will take whole or a subset of this model and bring it over to FIS-models. This subset can possibly be further partitioned. A similar process takes place on the FIS and the FCIS level as on the EIS level, before a CIS is created based on the final FCIS-model, and manual procedures based on the FIS-model is externalized, and this is all committed to (part of) the organization, resulting in a buffered transition of the COIS. One can also perceive a situation where several CISs for different parts of the organizations are created and committed to the different parts of the organization. In any case, the actor models should be updated to show which parts of the new CIS that are accessible by which users, using detailed support relationships. The overall pre-integration strategy should be made in early project planning, but should be updated as found appropriate during the project.

In addition to the creation of a CIS, the knowledge of the participants should have changed as part of the project. In the conference case the developer group learned about the arrangement of professional conferences in addition to conceptual modeling and specific technology such as Ingres DBMS and WWW. Both AS and especially JK have learned about these same technologies. In addition, and equally important, JK has learned a lot about the arrangement of professional conferences, thus one do not only have the potential for mutual learning [185] where developers learn about the application area, and users learn about new technology, but also for organizational learning, where different users learn from each other as part of participating in the project.

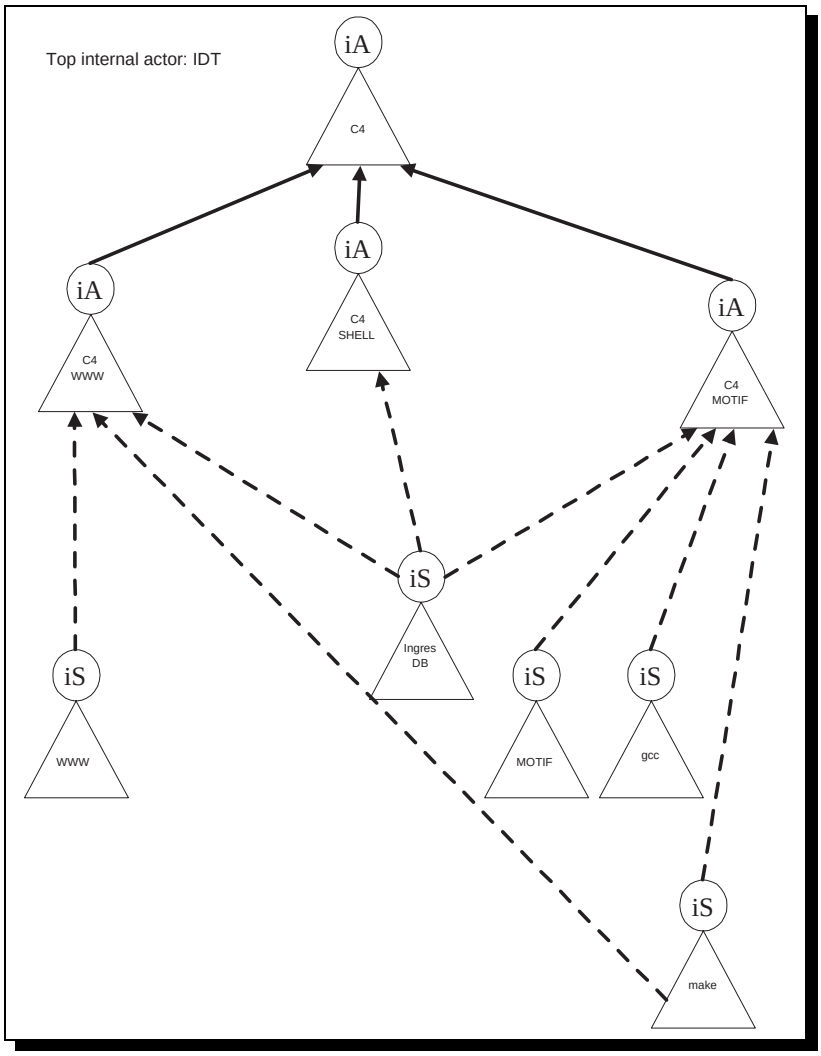

Figure 10.18: Actor model of the C4 application system

10.2.3 Conceptual Models in Usage Support

Due to the large number of direct (and indirect) users of many application systems, the number of users which can participate extensively in the devtenance projects is limited. Thus before committing the updates of the COIS to parts of the organization, one need to perform training of the direct users. There are three main reasons for including the usage of conceptual models in this training, and use the developed conceptual models in the process of producing user-documentation:

- The version of the application system that is to be committed to the organization is in most cases not the final system. Thus, to enhance future participation in the evolution of the COIS, users should have a good understanding of how the current COIS function. When a maintenance project is undertaken at a later stage, it should take less time for all participants, also those not taking part in projects where previous versions of the application system where developed, to understand the current situation as it is externalized in the existing CIS, and to be active in the evolution of the application system. When using the conceptual description during training, one can use the techniques for improving comprehension of the models, such as filtering, execution, and explanation generation to enhance knowl-

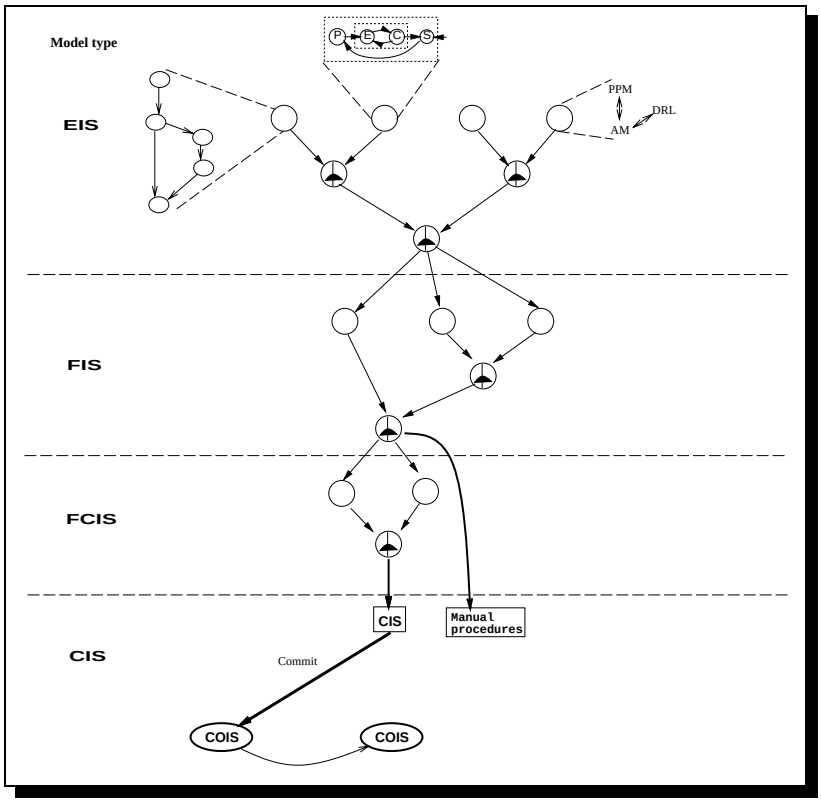

Figure 10.19: Conceptual modeling in a devtenance project

edge among the users about how the system function in a better way than is possible when the end-users only have a user-interface to relate to.

- If a large part of the resulting application system is generated directly from the conceptual models and/or traceability between the application system and the conceptual models are well supported, it should be possibly to go "behind the scenes" as described above following the items as they travel along flows in the process model also in a normal use-situation, applying the more advanced techniques for comprehension.
- If a system devtenance methodology is adhered to in the organization, also other projects on other areas of the CIS-support will use some of these conceptual modeling languages. Thus the training received to comprehend such models in connection with this project should cater for a shorter learning period if the user is to participate actively in the development of another system.

When including knowledge of conceptual modeling in the user-training, one should concentrate on reading skills, i.e. it is not necessary for the users to be able to write models in the languages at this point. One should focus on the process modeling language, since this is the focus of the main comprehension support techniques such as execution and explanation generation. One should also learn the main parts of the data and rule modeling languages, but only go into the details of these on a need-to-know basis. The user-model for these users are also developed based on the experience they receive in training

If complete code-generation and traceability is possible, one can imagine that the explanation generation facilities can be used actively also during use of the system. As discussed briefly above, it is not to be expected that this will be so in all cases due to the large diversity of sup-

porting software and hardware actors. When one *is* able to produce the run-time system directly based on the conceptual models for example by using the Tempora run-time system, the following support can be envisaged:

When completing the rule-hierarchy down to rules for the executing processes, the user can have the possibility to interrogate not only what the system do, but *why* the system do this based on the motivation from a higher level business goal. For this to function in a satisfactory way, it should be possible for the user to use the explanation generation facilities towards a database that mirrors the live database, onto which one can perform *what-if analysis* that gives a similar behavior as would have been observed when using the application system. Suggestions for annotating the application system with easier understandable artifacts to assist the help-facilities of the application system are given in e.g. [226]. The difference in our case is that one reuse the models that have been developed as an integrated part of a devtenance project.

If the users experience breakdown in the usage, it should be possible for them to refer to the conceptual model in COISIRs, and not only to the place in the application program where the error occurred. It should also be possible for them to access the actor-model indicating which internal social actor supports the application system that they are currently working with, or finding out which other actors are supported by this part of the system.

Also when not having close links between the conceptual model and the application system should an integrated system for the creation of COISIRs be present, thus it should be possible for the users to easily specify system investigation reports and send them to the front-end of the CIS-department. Preferably, support for the development of COISIRs can be included in application system in a way similar to how help-text is integrated, thus all the user need to fill in is a problem-description, perceived urgency and type if known. This can then be sent to the front-end of CIS-support automatically.

In the conference case around 60 COISIRs were submitted, including requests for both corrective and perfective maintenance, in addition to general questions.

Even if it would be possible, it is not to be expected that most users will perform detailed investigations of the conceptual models by the use of explanation generation facilities themselves, but for the front-end receiving COISIRs, (and possibly also the back-end) this kind of functionality can be useful when prioritizing and classifying the request. Based on the links from the actor-model to the other models and traceability between models, it should be possible to find out in detail the part of the application system that is supporting the individual actor in case this is different in different parts of the organizations. The actor-model could also be linked dynamically to the executing application system, and be updated as the database is updated with new instances of the entity-classes that describe roles that are filled by the actors.

In addition to the reception of formal COISIRs, one should be able to receive such request also in a more informal form, e.g. through phone calls, causal conversations, memos, or e-mail. If the request requires modifications to the applications, the action to be taken and the time the change will be committed to the part of the organization which the requesting user is a part of should be communicated to the source of the COISIR. Urgent problems are attacked directly by the back-end, changing the conceptual models underlying the application system if necessary, and regenerating the necessary parts of the application before committing the changes to (parts of) the organization and updating the request base.

When a problem is found in the supportive software, one can use the actor-model to identify who supports this actor and the front-end can have direct contact with these. All COISIRs, no matter who is the source and the form it arrives in, are processed in the same way. The set of submitted COISIRs give an indication of the perceived semantic quality of the model in the

implemented application system.

In addition to the normal log-messages, the implementation of the deontic operators will give additional data to the front-end reporting on the overruling of recommendations, attempted security-violations, and performance of discouraged actions, and the back-end can use these in the assessment of changing the modal status of rules in later versions of the application system.

Finally, one should investigate the use of user-groups and focus-groups [162] within the organization also for customized solutions, and in this connection be able to utilize the conceptual models of the portfolio.

10.2.4 Conceptual Modeling Based on Existing Conceptual Descriptions

Here, a project is started with a populated specification base and a set of unsatisfied request in the COISIR base. Except from that, most things are similar as in a "development" project describe above, and is not repeated here. On the other hand, additional constraints exist because of the existing conceptual description.

In the case of a new project as part of incremental development, a model of parts of the current situation already exist because of the CIS-model, and thus can the perceived model of the situation be produced more easily based on this, and partly also the model of the perceived future situation. One should take into account the requests for changing the existing system and start from this together with the overall model of the future total information system of the part that is within the new application boundary.

The audience of this project is often different from the one during earlier developments. In the conference case,

$\mathcal{A} = \{A_1, A_2, A_6, A_{10}, A_{12}, A_{13}\}$

where A_1 = AS, A_2 = JK, A_6 = KPH, A_{10} = FVL, A_{12} = ISDO2, and A_{13} = PPP.

The main audience-members at this stage were: A_2, A_6. In addition COISIRs were received from some of the PC-members testing out the WWW-interface for the reception of reviews.

The changes to the system were primarily based upon the already developed models in the first project. For instance the situation depicted in Figure 10.14 is largely the same on the high level, the difference being that also process 5.2.2, 5.2.3. and 5.2.4. were detailed further and implemented as part of the application system.

On the other hand, new needs might also be discovered. In the conference-case it was decided to use discussants on the accepted papers, being largely motivated from the goals of having a high quality of the conference, and to get more conference participants. This spawned a set of new rules, and the need for a new PPM as depicted in Figure 10.20.

In addition did this result in the changing of two of the other process-models and one additional relationship was added to the ONER-model linking the entity-classes "participant" and "paper" with the relationship-class "discussant". Also the actor model was updated accordingly as discussants were recruited.

Since different alternatives about how different situations can be tackled are retained, including the argumentation-process behind the choice of the existing solution, one have a broader set of solutions as a starting point for new discussions.

A simplified extension of Figure 10.19 including a follow-up project is given in Figure 10.21. As we see, the newly externalized CIS and manual procedures are taken as one outset, that are used together with the earlier developed EIS-model, and based on this a new model of the existing reality as it is perceived can be created. A similar process of sub-setting, splitting and merging as for the development project can be conceived, but we have not indicated this in the figure. We

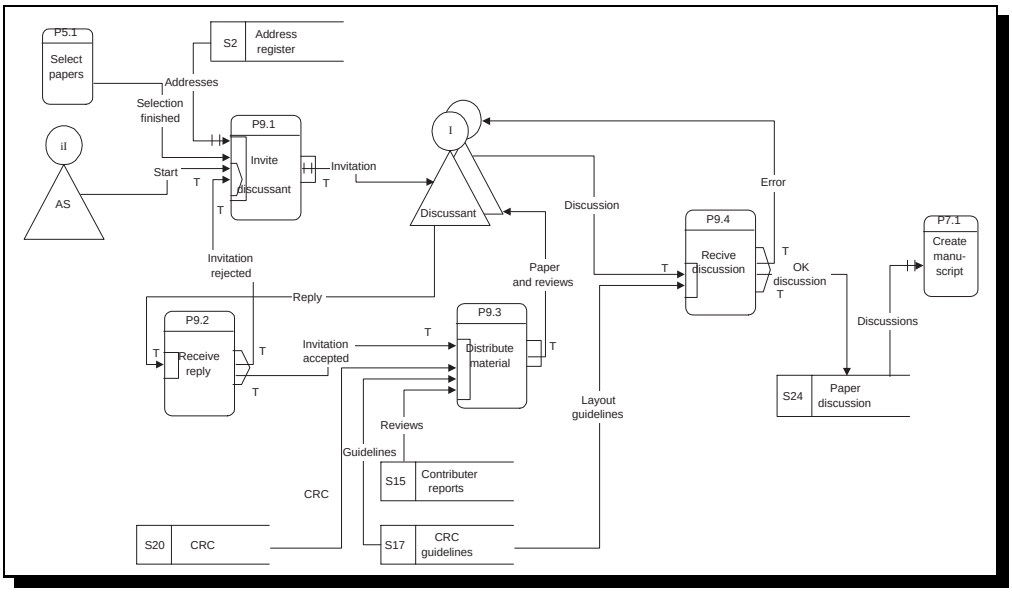

Figure 10.20: PPM describing invitation of discussants

have included existing COISIRs as being taken into account on FIS and FCIS-modeling though. When the new version of the CIS is finally committed, this is a new buffered transition of the COIS.

There are several other cases where a devtenance project is performed on the background of existing conceptual models. If an organization wide model has been created, this can be regarded as one of many possible models of the current situation, and one should go through the process for improving the social quality of the models, which might mutually influence both the organization wide model and the model for the limited area that is taken into account in the devtenance project. In the case that this part of the enterprise-model is also referred to in another application system model, one must assess the need for updating also other application systems e.g. if these systems refer to the same data. In this way one is able to keep the data models sufficiently consistent on the portfolio-level.

Similar treatment is given models developed as part of an application framework, although these are not changed themselves since these are usually developed and maintained by an external actor.

In the case of a replacement system being installed, essentially the same problem appears, although the actual design step might have to be different, i.e. we go into the methodology in the second phase developing alternative future informations systems in a problem-oriented manner. In this case it is especially useful to be able to track the design decisions made in the earlier project which are dependant on the supporting software and hardware actors. When the replacement system is made by putting together functionality of two or more existing systems, integration of the conceptual models for the existing systems can be performed as has been discussed earlier.

When further developing a system based on the conceptual models, one ideally should be able to regenerate the new system automatically, effectively replacing the existing system. On the other hand, one do not want to retest the complete specification, thus something similar to regression testing should be performed based on the changes done to the conceptual models. The close integration between the models in different languages can be utilized for this purpose, since it would be easier to track the side-effects of changes. The conversion of data from the existing

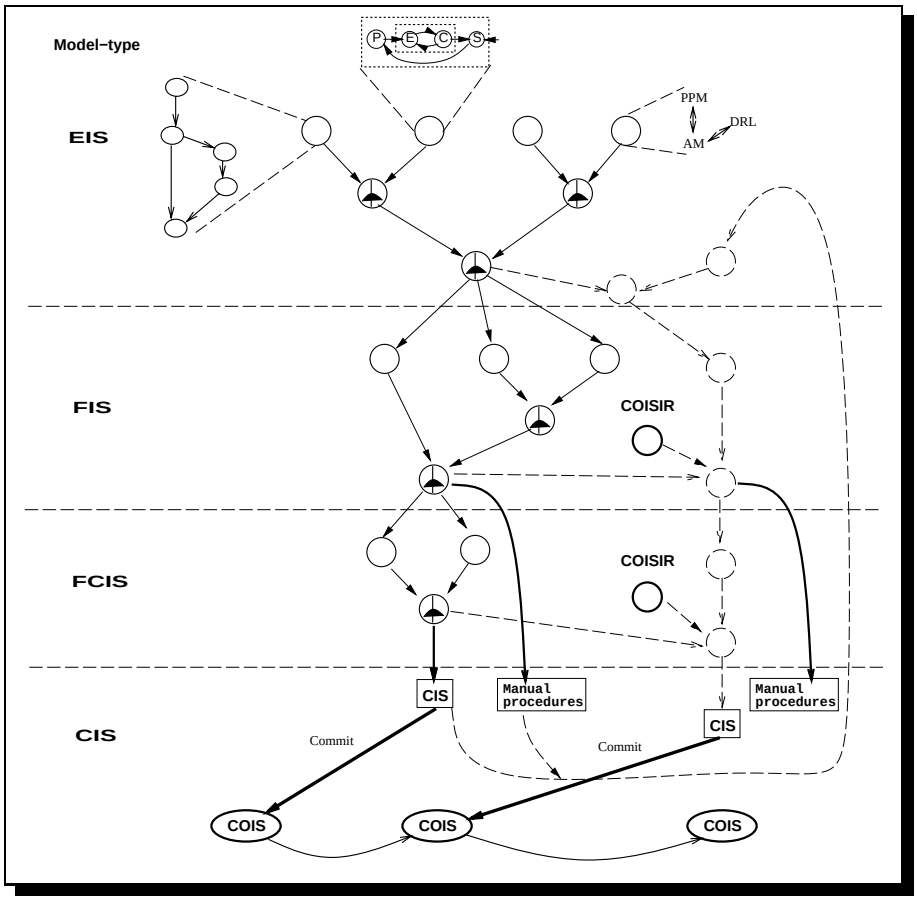

Figure 10.21: Conceptual modeling in a follow-up project

database to the new database to be constructed should also be possible to support by the knowledge of the changes that are done to the data model and additional optimalization. A detailed discussion of these matters are regarded as beyond the scope of this thesis.

10.3 Chapter Summary

We have in this chapter presented our methodological framework in more detail, by showing how a set of conceptual modeling languages can be used within system devtenance. This does not mean that one needs to use the above mentioned techniques and modeling languages in the way presented here to apply the overall framework. On the other hand do we hope that we in this chapter have given important arguments for the feasibility of the suggested approach to CIS-support in organizations.

In the next chapter we will perform an overall evaluation of both our methodological and our conceptual contributions as they have been presented in the last three chapters.

Chapter 11

Evaluation

Based on the last three chapters, we will evaluate the new methodology and the extended set of modeling languages based on the classification of a methodology presented in Chapter 4 and the criteria for model and language quality from Chapter 6. Comparisons will also be performed towards the current state of the art in methodological framework and conceptual modeling as presented in Chapter 4 and Chapters 5 and 7 respectively. When it comes to rule, role, actor, and speech act-modeling, we will also compare to work not described in the state of the art chapters where appropriate.

11.1 Classification of Methodological Framework

System devtenance can be given the following classification:

- Weltanschauung: Constructivistic. This view has been important for the specification of guidelines for a more detailed methodology. Especially it has resulted in an emphasis on stakeholder participation. It has also been influential on the approach to conceptual modeling including the framework for quality in conceptual models and modeling languages. Also when performing the conceptual extensions we have had this in mind, both regarding the modeling of actors including the indication of the source of modeling statement, who institutionalize rules and roles, and the inclusion of deontic operators both within and between rules, capturing the deontic judgments behind the models. The extended coverage of speech acts is also believed to be helpful when modeling from a constructivistic point of view. The methodology is also partly influenced by the constructivistic methodologies described in Chapter 4 e.g. on the use of CATWOE-analysis.
- Coverage in process: Both development and maintenance are covered in an integrated manner. Usage is also addressed although not in the same detail by a description of how one can organize the CIS-support to handle e.g. COISIRs and emergency changes. It is primarily aspects that are related to conceptual modeling which have been described in detail in the thesis. The capturing of deontic judgement are believed to be especially useful in connection to maintenance of application systems, since it can support the sense-making of existing systems, and hopefully help in avoiding premature closure as the situation that the application system support changes.
- Coverage in product: The whole portfolio is meant to be supported, although most of the suggestions in this thesis are geared towards the support of single application systems. The conceptual framework is extended with actor-modeling to support modeling on a portfolio

level, but more work needs to be done on specific aspects of portfolio-support. The inclusion of links from operational to high-level rules might also be helpful seen from the point of view of portfolio support.

- Reuse: Generative reuse through model and code-generation and reuse between phases and projects are supported. The view of maintenance as reuse-in-the-large is adopted from Basili [21]. Suggestions for supporting compositional reuse are made based on modeling of technical actors, although much work is still to be done on this area.
- Conceptual modeling: The methodological framework is based on active use of conceptual models written in integrated languages covering different perspectives. A more detailed evaluation of the conceptual framework is given below.

Compared with the classification of PPP from Chapter 7.4, we see the largest difference in the Weltanschauung, and the attempted coverage of process and product. Looking back at Chapter 4, we find no existing methodological framework in the literature with this mix, of as argued there, beneficial aspects. The major weakness of the approach is obviously its immaturity, both in the lack of comprehensive tool-support, and lack of practical experience with the approach. It is indicated in the thesis how the major extensions of the conceptual framework can be supported with the PPP CASE environment. The experience with the approach is so far limited to case-studies.

11.2 Evaluation of Language Quality

We will here primarily look upon the improvements and potential problems with the extensions of the modeling languages as presented in Chapter 9. We will not repeat the discussion of PPP from Chapter 7.4 on areas that we have not attempted to address. These comments and critiques are still valid. Some additional critique of the existing languages that has appeared as we have worked with our extensions, will be described briefly.

11.2.1 Domain Appropriateness

The overall support for the structural, functional, and behavioral perspectives are as before with the inclusion of temporal rules. From an object-perspective, the appropriateness is slightly improved, by the object-based nature of the actor-modeling language. The support of the actor and role perspective is also improved through the extended actor language. Both human and technical actors can be represented, and also how they are related to each other and to roles, how they communicate and how they support other actors and roles in addition to processes, flows, and stores in the PPM. The communication perspective is further supported through the addition of the possibility of extending flows with the speech acts that sending an item on this flow implies. With this extension in addition to the other extensions in the use of actors and roles in PPM, it has been illustrated in Chapter 9 how the extended language can express similar situations as can be expressed in action-workflow [221], ABC [71, 314], and SAMPO [14] when it comes to the modeling of speech-acts. A difference is that we have made it possible to integrate the speech-act modeling closely with process-modeling which can be both positive and negative. Whereas the extended expressiveness is positive, a negative effect is that one might not as easily concentrate on the speech-act modeling as in the more specialized approaches. Another aspect which are supported specifically in SAMPO and COMMODIOUS [136] is the creation of contracts. Contracting can be indicated in our approach by applying rules added to the speech acts, but this is not as immediately intuitive. Another approach combining speech acts and deontic operators are

reported in [75], where more work has been done on indicating how deontic rules can be implied by existing power and authorization-structures and the performance of speech acts.

Comparing with the critique of speech act modeling summarized in [65], we note the following:

- The sending of an item can be combined with several speech acts. The item might be composed of many "utterances", thus in principle having the possibility of modeling many-to-many and not only one-to-one mappings between utterances and illocutionary acts.
- One do not need to model the illocutionary acts involved when sending an item, nor making the system based around conversations if this is not found beneficial.
- One is not forced into a fixed conversation structure, although one can support modeling according to a given structure if this is perceived to be beneficial.
- By the addition of rules, one is opening up for giving permissions, and having recommendations, and not only obligations in connection to the speech acts.
- One can easily integrate speech-act modeling with other modeling perspectives.

In the terminology of Wegner [326], the actor-language applied to technical actors is object-based, but not object-oriented, since one are not saying anything explicit about how actors are structured. As in e.g. OORASS [267] roles may be played by any number of objects (actors) and an object (actor) is also able to play many different roles. In our approach, we have only enabled this for actors and not general entities. The inheritance mechanisms between roles and actors are less strict than in traditional object-oriented modeling. One should look closer on the inclusion of state-transition modeling for actors and other entities.

When comparing with ALBERT [78] we find that most of the same notions can be used in our framework, but the strength of ALBERT compared to our work is in the specifications of actor dynamics. Our actor-modeling language on the other hand is more expressive when it comes to relationships between social actors, and communication between actors.

Compared with traditional organizational modeling, we claim that we with our mechanisms can model any form of organization-structure. This is also the case in RIN [288], but RIN has an explicit presentation of the interaction between roles. Although we can model similar situations in the extended PPM as in a RIN, the role- perspective might be better to use than the process-perspective when wanting to concentrate of roles and not processes. The modeling of actors are better supported in our approach than in RIN.

The dependencies described in Yu and Mylopoulos [344] can be specified in our approach using the support link, indicating if necessary the detailed form of support. One can also use PPMs to indicate the more detailed support relationships more indirectly, although specific dependencies are expressed more economically in AD. The Agents-Roles-Positions modeling language have the same expressiveness as basic actor-modeling.

Static aspects of an actor or role can be expressed through the link to the ONER-model, whereas dynamic and temporal aspects are not as explicitly expressible. That an actor is given or looses a role can only indirectly be represented through the link of the role to the ONER-diagram that is linked through a store in the PPM, which in turn is updated by a process in the same model. Other relationships, such as authorizations, can not be updated dynamically in the same way, something which is supported in e.g. [75, 188]. It is the lack of support of dynamic and temporal aspects of actors that are the main weakness of the suggested actor-modeling language in our point of view.

The rule perspective is supported much better than in the previous version of PPP, through the DRL language. Both single rules and relationships between rules, and most importantly, links between rules and actors, roles, and the other modeling languages can be expressed. Through the rule-language and the improved possibilities for specifying the behavior of timers, the temporal

expressiveness is improved, although this is not yet fully exploited.

There are many approaches combining rule languages with other modeling languages. Tempora, which was a starting point for our approach, combined rules with structural and functional modeling languages. The same has been done in [320]. ERAE [119] coupled rules in temporal logic to diagrams showing entities, relationships and events. Rules have also been combined with Petri nets [188, 292]. Coupling of rule-based and object-oriented approaches can be found in e.g. [264], and couplings between rules and actor/organization-models in [78, 344].

DRL is not an application of deontic logic, but of deontic operators. The main motivation for this limited approach is to avoid some of the paradoxes of standard deontic logic and yet obtain a more expressive language than first-order temporal logic. Few of the modal logic approaches to deontic logic are able to deal with recommendations. Those that do this e.g. [154] are not taking into account an underlying historical database, and mixing it with a temporal predicate logic. Few of the existing applications of deontic operators within computer science use a standard modal logic semantics for the operators [328]. This indicates that other semantical interpretations than a modal logic one can be useful. On the other hand, we note that Ryu [273] claims to be able to address the traditional paradoxes within a model logic framework by applying an extension of the principle of specificity. We would like to investigate if this extension also can be useful for us in the future.

Comparing our framework with existing approaches using deontic operators, we notice the following: Even if many acknowledge the need to separate rules of necessity and deontic rules [46, 329] none of the approaches that we have come across introduce a third level dealing with recommendations and discouragements, even if these notions have been discussed in deontic logic and are regarded as necessary for sufficient expressiveness of a language by among others Maibaum [212]. Approaches that use other operators than **O, P, F** do exist. Lee [188] introduces waivers, which are defined as $\mathbf{W}\psi \leftrightarrow \neg\mathbf{O}\psi$. Maibaum [212] introduces *Preference* ψ meaning that there exists at least one action other than ψ which is permitted. More interesting from our point of view is the work based on Hohfeld [3] which include various nuances of the standard deontic operators of permission, obligation, and prohibition as well as other notions like right, duty, no-right, privilege, power, liability, disability and immunity. Another inspiration for further work is the operators proposed by McNamara [220]. Worth mentioning is also that many newer approaches to deontic logic indicates both the bearer and the counter-party of the rule (see e.g. [131]), whereas we indicate either the bearer or the counter-party explicitly, but not both. We can in addition indicate the institutionalizer of a rule, but this is not necessarily the same as the counter-party of the rule.

In our approach, we use several kinds of operators, i.e. both modal, temporal and deontic. This is also found in other works, for instance LLD [217] and in the work of Maibaum [212] . Other examples are the mix of epistemic and deontic logic in [25], and the linguistically motivated language of Dignum et al. [73]. On the other hand, these approaches are not linked to an extensive set of graphical modeling languages.

Compared with the rule-relationships of Tempora, which supported necessitates, obligates, and recommends relationships, also the additional relationships of *permits*, *discourages*, *forbids*, and *excludes* are included. The relationships are better specified, and *or* and *and-nodes* can be used.

To our knowledge, no one has applied the five deontic operators together with exception and necessitation operators in a goal-hierarchy, even if similar notions may be expressed in an ad hoc manner in a goal-model such as the one used in the ABC method [333], or in a typed hypertext system such as HYDRA [262]. Those addressing motivation-hierarchies seldom mention exception-hierarchies and vice versa. Comparing with the languages of goal-oriented requirements engi-

neering, the relations proposed by Feather [87] between rules and policies can be expressed in the following way: *supports* can be expressed by *obligates* or *recommends* , *impedes* can be expressed by *discourages* or *forbids*, and *augments* can be expressed by *permits* or alternatively *necessitates*. *subgoalof* relationships can be expressed by using *and-nodes*. Subgoalof-relationships are often used in projects using military standards, see e.g. [58]. In the HOQ-technique from organizational development [200], one might also indicate similar notions as us between goals and rules although in HOQ one take a more quantitative approach. Comparing with the approach of Mylopoulos et al. [55, 232] we find that *obligates* and *recommends* are parallel to the *sub* and *sup* relationships although with a different semantics. We have no possibility in our approach to express the semantics of the sup-relationship. Similarly, *discourage* and *forbids* are parallel to the *-sub* and *-sup* relationships. *Or-nodes* and *and-nodes*, also supported in [9, 297] and [79] for the specific treatment of different viewpoints are supported. The more informal *related-to* relationship is parallel to the *und* relationship of Mylopoulos et al. Also they have the possibility to annotate relationships with arguments.

As pointed out by Jones [153], there is often a need to distinguish what is required per se from what is actually done to detect violations of rules. These situations often lead to so called *secondary obligations* stating what is obligatory to do in cases of violation of an obligation. Deontic operators are often felt to be appropriate to handle such cases. In our approach, secondary obligations can be represented for instance by using timers in the process models and by linking action rules to processes. Since the support of secondary obligations is often indicated as the main reason for using deontic operators by many, we will show how this functionality and also the related areas of bounded obligations [86, 163] and permissions with accompanying obligations and potential sanctions [224] can be expressed in the new combined approach. We will use the following short case-description taken from a library case study:

> A user must return the book within 28 days of borrowing it. If a user fails to return a book before the due date, then a fine will be issued and the due date on the book will be extended by seven days. The fine must be paid within seven days of issue. If a fine is not paid on time, then the borrowing rights of the user is blocked and the fine increased. The borrowing rights are only returned when all fines are paid and all books returned.

A possible PPM for this situation utilizing delays are shown in Figure 11.1

The ONER-scenario is not shown, but will basicly consist of two entity-classes, *books* and *customers*, and a relationship-class *p1 loan pn* between the entity-classes. The attributes of *customer* will apart from traditional attributes such as *name* and *address* include the attributes *borrowing_rights* and *dues*. Some of the rules for the example are presented below:

- Process P1 “Register borrowing”

```
when loan_request(Customer_number,Book_id)
if   customer.Customer_number has borrowing_rights = ‘‘OK’’
then it is permitted for customer
           customer has customer_number.Customer_number
                        loan book has book_id = Book_id
           and loan(Customer_number,Book_id)

when loan_request(Customer_number,_ )
if   customer.Customer_number has borrowing_rights <> ‘‘OK’’
```

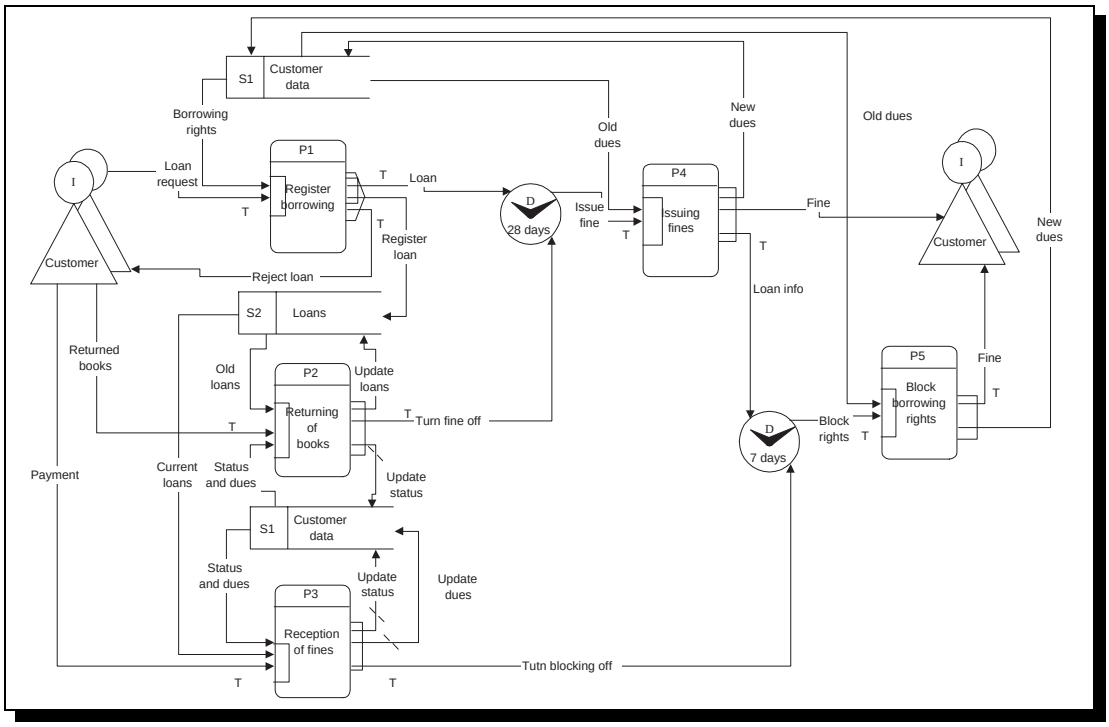

Figure 11.1: PPM for library example

```
then it is  obligatory for the system
reject_loan(‘‘Your borrowing rights are temporarily removed, please
pay your fines and deliver the books you have borrowed’’)
```

- Timer D 28 days:

```
when 28 days since loan(Customer_number,Book_id)
if not turn_fine_off(Customer_number,Book_id)
since  loan(Customer_number,Book_id)
then issue_fine(Customer_number, Book_id)
```

- Process P4 “Issuing fines”

```
when issue_fine(Customer_number, Book_id)
if customer.Customer_number [has dues.Dues
                             has address.Address
                             has name.Name]
then it is obligatory for the system
customer.Customer_number has dues = Dues + 100
and fine(Name,Address,100) and loan(Customer_number,Book_id)
```

The rest of the rules follow a similar pattern and are not shown.

Our inclusion of deontic operators in DRL has made it apparent that there is a weakness in the specification of e.g. cardinality-rules in the ONER-language and rules captured in a PLD, since it is not possible to indicate the deontic modality of these rules.

The new aspects (roles, rules, and actors) are general, but can be specialized. The composibility of the overall approach is improved. The languages are also very flexible in precision, and both vague and precise statements can be made and interrelated.

11.2.2 Participant Knowledge Appropriateness

PPP is based on much-used modeling languages such as ER, DFD and rules, although extended to increase the expressiveness. For those familiar with this basis, the PPP languages as such should be rather simple to learn and understand.

The basic phenomena of actors and roles do we also believe to be generally easy to grasp for most person being familiar with e.g. organizational charts. Main parts of the actor-modeling language has been tried out by Noreide in [235] in addition to in our own case-studies, with no reports on the language being difficult to apply or comprehend. The rule-hierarchy as a way of detailing rules do we also believe to be generally easy to comprehend. In addition is it possible to express in these variety, conflicts, and inconsistencies explicitly. However, the formal DRL-language suffers from many of the weaknesses of standard rule languages, and we would expect similar problems with applying this as was found in Tempora. We have addressed this by only applying the detailed language in special circumstances, by being able to use PLD's instead of DRL-rules to describe process logic, and to link the rules to the other languages, to higher-level rules, and to actors and roles.

When it comes to speech-act modeling, we would expect that for some users the linguistically motivated parts will be difficult to understand and apply as reported in [41], but we need to experiment more with this issue, especially the combined use of process and speech-act modeling.

The differentiation between main classes of actors are made through the use of natural language specific codes. Natural language specific codes are also used for the short form of deontic operators, and the illocutionary points. As also discussed under the treatment of PPP, this is regarded as a minor problem that could be supported in a modeling-tool by changing the external representation of these phenomena according to the preferred natural language of the user of the tool.

11.2.3 Participant Interpretation Enhancement

The previous unclarity as described in Chapter 7.4 between functions, processes, and methods has been removed. External agents are divided into two phenomena classes, actors and roles, which symbolically might still be too close. It is possible to have different external representation of different types of actors e.g. in an iconographic form with the proper tool-support. Although adding much in the expressive power, the number of major phenomena classes is not very much higher than before. On the other hand, this might have resulted in overloading of especially the abstractions 'actor' and 'role', which can be used to indicate very different phenomena such as individual persons, organizations, and application systems. The problem of over-loading can also appear on the actor relationships and support links.

The agent symbol is rather complex compared to the other symbols, but because of the limited use of the symbol this is not looked upon as problematic.

By the changed use of actors and roles, also these might have a large degree in the diagrams, thus potentially increasing the problem of e.g. crossing lines in PPM. This can be helped by symbol duplication as traditionally done in process-modeling.

Both actor and rule-modeling include abstraction mechanisms, but good visual abstraction mechanisms for these models are not implicit in the languages, but must be supported explicitly in the tool support.

When annotating actors and roles on relationships between actors one can often apply the same symbol to several relationships, thus achieving expressive economy of the external representation.

Additional negative aspects with ONER, PPM, and formal rules can be found in the evaluations given in Chapter 7.4 and Chapter 7.2. Some of the negative aspects of PPP are addressed as indicated above.

11.2.4 Technical Actor Interpretation Enhancement

The modeling languages have a defined syntax and operational semantics for the relevant parts of the languages. It has been indicated how these can be used in the creation of prototypes and explanations in Chapter 9.3 and Appendix D. Little has been done to improve executability in the approach for efficient execution. By applying transactions as in Tempora, we hope that also executability can be improved. The deontic operators can indicate priorities at run-time. If the load on the system is too large to execute all enabled rules, obligatory actions will be given priority to recommended ones. If more than just two levels of priority should be requested, this can be supported by providing recommendation with a numerical parameter. The use of this is currently not investigated in detail in the approach.

11.3 Potential for Creating Models of High Quality

- Physical quality: We have indicated how the meta-model of the tool can be extended to include the support of our extensions, and this can be included in a future version of the PPP tool. Configuration management support should be re-assessed and extended based on the suggested way of working and the overall framework.
- Syntactic quality: The modeling languages are formally defined and thus opens up for both error prevention and error detection. Further functionality to support this should be implemented in the PPP-tool.
- Semantic quality: Tools for updating models are currently developed, and it is felt that the proposed structuring mechanisms of the modeling languages help in the task of changing a large specification and keeping track of changes. Algorithms for advanced consistency-checking of both the process and data-model have been devised and is incorporated in the tools that are being developed. However, consistency checking offers only limited support when it comes to establishing the validity and completeness of a specification. The most important contribution to quality assurance in the extensions is considered to be the inter-rule operators, with their possibility of structuring the rule-base and providing motivations for low-level rules. This enables the following check questions for
  - Validity: Are there any low-level rules which are not obligated or recommended by higher level rules?
  - Completeness: Are there any high-level rules which do not have corresponding low-level rules, or for which the low-level rules registered so far do not seem sufficient?

  The driving question technique based on the interconnection of the languages is believed to be useful for improving perceived semantic quality.

The deontic operators enable the indication of priorities among requirements.

- Pragmatic quality: Several different techniques for potentially increasing the pragmatic quality of models have been described, and are partly integrated, such as explanation generation, execution, and filtering. Also primitive simulations can be performed. We have indicated in the thesis how these mechanism can be extended to take into account our extensions of the modeling framework. Especially interesting is the suggested extensions of the explanation generation that would make it possible to use higher-level rules in the rule-hierarchy when creating the explanations, providing the motivation for why the systems is behaving as it is. On the other hand, the techniques are only partly integrated, thus even if the separate approaches have been shown to be among the state of the art within their fields, more work must be performed to be able to use the techniques in concert. Browsing functionality between models should also be extended. Layout modification based on aesthetics is currently not supported.
- Social quality: It is possible to register explicitly which rules are conflicting, by means of the *discourages* and *forbids* relationships, and through the use of or-nodes. In this way, conflicts can be registered also at levels where rules are informal and hence do not lend themselves to consistency checking. The resolution of conflicts, of course, is still a process which cannot be automated, but at least it helps to be able to register conflicts in a systematic manner. The approach is integrated with IBIS like techniques to support the argumentation-process. Special to our approach is the linking of actors that support the goals to the actual goals and relationships between goals, enabling a explicit modeling of different viewpoints, and easy discovery of issues for debate. By retaining the argumentation and preliminary pruned branches in the goal-hierarchy, one also get a natural way of keeping track of design-decisions, thus enhancing traceability [109, 266].
  Although briefly discussed in this and earlier work on PPP, there is not as yet developed any satisfying techniques for supporting model integration within the approach.
- Knowledge quality: Indication are given in the methodology for ways of doing stakeholder identification, participant selection, and participant training.

Finally, it is indicated in the methodology ways to use the existing and perceived techniques as part of an overall methodology.

11.4 Chapter Summary

We have in this chapter performed an overall classification of the approach suggested in this thesis, and in addition performed an evaluation of the conceptual framework in particular according to the framework for assessing language quality, and potential for creation of models of high physical, syntactical, semantical, pragmatical, and social quality.

On a high level of abstraction, the framework covers the important aspects discussed in Chapter 4, but it is only the aspects connected to conceptual modeling which are described in detail in the thesis. The overall conceptual framework can be used for modeling according to all the main perspectives discussed in Chapter 5, but there are not surprisingly room for improvement especially regarding participant interpretation enhancement. Most aspect of model quality can also potentially be supported, although this is dependant on the implementation of additional tool-support and further practical experience according to the guidelines of Chapter 10 to create a way of working that will support the creation on conceptual models of high quality. One important area that is not supported in depths in the existing literature on PPP are model integration, and

this should be investigated further.

In the final chapter we sum up what we regard as being the main contributions of the thesis, and the main areas for further work within the suggested approach.

Chapter 12

Concluding Remarks and Future Directions

'Would you tell me, please, which way I ought to go from here?'
'That depends a good deal on where you want to get to' .

Lewis Carroll

In this chapter we sum up the perceived achievements of this work and mention some aspects that should be dealt with more extensively, giving directions for further work.

12.1 Major Contributions

We have in this thesis presented a methodological framework for CIS support in organizations. We have specifically investigated the use of conceptual modeling within such a framework, based on social construction theory. The major contributions of our work are regarded to be the following:

- We have performed a survey investigation on development and maintenance practice in Norwegian organizations giving partly novel results, especially on the distinction between functional development and functional maintenance. The results of the investigation being presented in the thesis forms together with results from similar investigations performed by other researchers the main empirical background for the overall suggestions regarding a methodology for CIS support in organizations. Results on different areas of the investigation have been or will be published in [170, 173, 175, 176, 181].
- We have combined conceptual modeling with social construction theory. Although also other approaches based on social construction theory have applied traditional conceptual modeling languages such as DFD and ER, it has to our knowledge not been attempted with a more comprehensive framework for conceptual modeling which cover many modeling perspectives in an integrated manner. We have in this connection extended an existing framework for the understanding of quality of conceptual models and conceptual modeling languages inspired by social construction theory. The quality framework has been published in [177] and is applied specifically for requirements specification in [178]. Process heuristics based on the application of the framework are published in [287].

- We have proposed a methodological framework for CIS support in organizations combining development and maintenance having support for not only the single application systems, but the whole application system portfolio. Other important aspects of the framework is participation based on social construction theory, reuse, and conceptual modeling. It is in particular illustrated how one might apply conceptual modeling techniques within the framework. These results are partly based on work earlier published [179, 197, 198], but also many new aspects are included.
- The conceptual modeling framework in PPP has been extended in several areas. Based on a taxonomy of actors, an actor-modeling language for the representation of both social and technical actors has been developed and is integrated in the conceptual framework. The conceptual framework is also extended with support for speech-act modeling. Support for rule-modeling has been extended based on work in Tempora. In addition we have included deontic operators in and between rules. The last extension is integrated with argumentation techniques. A presentation of the intra-rule deontic operators has been published in [180], whereas an overview of the inter-rule deontic operators including the linkage to an argumentation system has been published in [172]. The conceptual extensions have been integrated into existing techniques in PPP that can be used in the process of constructing high-quality conceptual models.

The feasibility of the approach has been illustrated as follows: The overall framework is partly based on the empirical indications that are given in both our investigation and investigations by other researchers, and on existing methodological frameworks described in literature. The conceptual extensions and partly how they can be applied within the framework are illustrated through several case-studies, and also build upon experience with the Tempora approach, which has many similarities with the approach that we have suggested, and on other conceptual frameworks described in the literature. A case study using some of the conceptual extensions has also been performed by others [235]. How to integrate the extensions with the existing approach for conceptual modeling in PPP has been outlined. Finally, the conceptual and methodological framework has been evaluated with respect to the framework for conceptual modeling, which although revealing some lacks and weaknesses, indicates that the approach can be extended to support the construction of conceptual models of high quality as part of systems devtenance.

12.2 Limitations and Future Directions

The following issues should be addressed in future work:

- A further integration of the proposed techniques, and an implementation of these in the PPP-tool should be performed. This should also include the further integration of the modeling of non-functional aspects such as user-interface and performance. Based on this, one should perform testing of the tools and the suggested techniques and refine these through extensive case-studies. The establishment of preliminary process-metrics could be started based on these case-studies.
- Techniques for model integration should be investigated in more detail, and integrated with the other techniques. This also includes investigating a better support of compositional reuse in the framework.
- One should look into more detail on configuration management issues, including the issues arising when also taking into account the evolution of the underlying CASE-tool, and not only the COIS, and additional issues arising when supporting the whole portfolio of an or-

ganization and not only one application system at the time.
- Reverse engineering from application systems using different supportive software and hardware actors to models in the conceptual framework should be investigated.
- Generation of production-quality code from the conceptual models towards different supportive software and hardware actors, especially middelware solutions, should be looked into. In addition to this should one also look closer into the support of testing of application systems made based on conceptual models, including regression testing to be used in projects which are based on existing conceptual models.
- Further organizational issues, such as project management, and extensions of (software) process modeling should be investigated and integrated in the overall approach.

Since the second of the above points addresses one area within conceptual modeling where the PPP-framework has proven to be insufficient, we will end the thesis with an introduction to ideas for an approach for model integration.

12.3 Model Integration in System Devtenance

Model integration is a general process where one based on several conceptual models create a single model. A general introduction to this area has already been given in Chapter 6, and we refer back to this for references to relevant literature in the field.

Through the thesis, it has been recognized that model integration techniques can be useful on many different areas of conceptual modeling, not only to enhance social quality of models. We have divided the overview given here in three areas:
- Intra-project integration.
- Inter-project integration.
- Inter-organizational integration.

12.3.1 Intra-project Model Integration

In this case, the model integration happens within a project, integrating models that are created specifically in the project. All the actors that are the sources of the models are supposed to be available. The languages that are used for modeling in this case are all part of the conceptual framework.

Several cases can be perceived:
- A set of models created by the same actor in different modeling languages need to be integrated. This typically needs to be done before integrating the views of the actor with models based on the local reality of other actors. To support this one can use the same links between models that are utilized for the driving-questions techniques. In addition, one should try to get the vocabulary consistent, but as part of building up the thesaurus also indicate the semantic distance between similar and dissimilar terms [161].
- Two or more models developed independently by different actors in the audience are to be integrated. Depending on the overlap of the areas of modeling, one should expect more or less overlap in the models. Generally, one should in this case rather look after similarities than differences. When the models are written in the same language, and using the same parts of the language, one can compare the constructs in the models, and in addition apply the thesaurus developed. An example of this kind of matching was given in Chapter 10, in the integration of the model of Figure 10.10 and Figure 10.11, and the discussion is not

repeated here. As indicated in work on model integration of structural models [94], the same situation can often be described using different modeling constructs. Because of this it might also be beneficial to compare the statements of the models and not the modeling constructs directly, but also here use fuzzy search applying the thesaurus information.
If the models are developed using different parts of the language, the comparison is limited to the parts that are used in both models, when looking at the component-level. When looking at the statement-level, there is no change.
If the models to be compared are developed in different languages, e.g. PPM and ONER, one can use the driving questions technique to create preliminary models in the other languages, and use these for the first comparison. Alternatively, one might compare on the statement-level directly also here.

- The models have a common predecessor model which they are based on. The models to be integrated can either be created by the same actor or by different actors. This area include the traditional merging of two variant models as discussed by Andersen [4], or the integration of viewspecs as discussed by Seltveit [282]. We can approach these cases in the same way. In both cases, one start out with a set of statements in the original model, and based on this adds and/or deletes a set of statements. When using a filter, one starts out deleting a set of statements from the original model, which are to be put back at a later point in time. The main approach in this situation can thus apply the explicit knowledge of the statements that are inserted or deleted as part of modeling.

12.3.2 Inter-project Model Integration

The models are developed in two different projects within the same organization. Examples are:

- A replacement system is made, based on functionality from one or more previously independent application systems which have been developed and maintained using conceptual models. These models can be expressed in both the same or different languages.
- A maintenance project, further developing an application system which have been developed using conceptual models in the case where one do not automatically reuse the whole existing conceptual model. The existing models can also have been created through reengineering.
- An enterprise-wide model has been developed, which model areas that are to be developed in more detail in a devtenance project. This is similar to the above situation.
- Several application system needs to access similar data, and one need to compare the structural models describing the data used by the different systems.
- Reuse of conceptual models developed in another context within the same organization.

In this case, additional terminological problems may appear. In the Sweden Post case for instance, what the Post was selling was called a 'product' by the accounting department, and an 'article' by the marketing department. The data associated with 'article' and 'product' was different although both refer to the same phenomena [282]. To address problems of this sort, a two-level thesaurus-structure, including also a taxonomy of application domains [104] might be useful.

To further illustrate potential problems in this area, we have compared the ONER-model supporting the paper-process both from the conference case (Figure A.28) and as it is modeled by Yang [340] based on the original case-description for an IFIP-conference (Figure 12.1), i.e. to simulate the comparison of an earlier application system supporting an area and the needs for a new one. The model of the IFIP-case also consisted of a separate scenario in connection with the practical arrangement of the conference, but this does not overlap with our model.

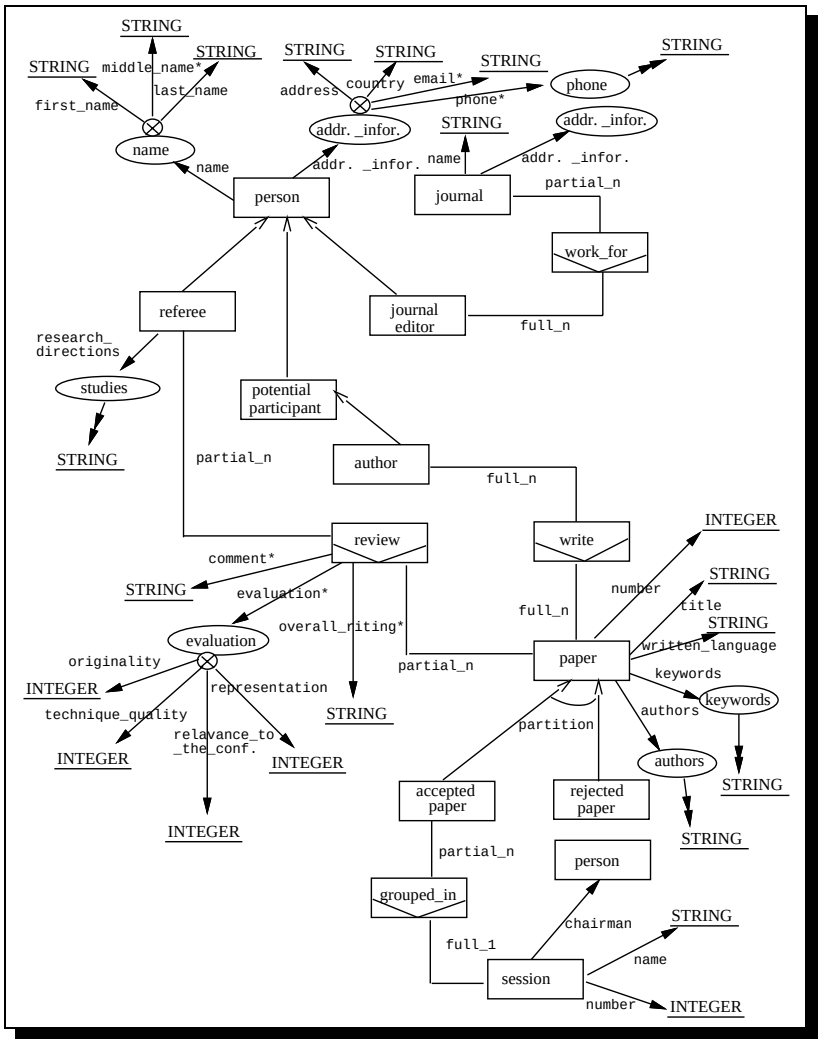

Figure 12.1: ONER-model for the traditional IFIP-case (From [340])

Comparing these models, we observe the following:

- There are some common entity-classes, e.g. person, referee, paper, and session.
- There are some common types, e.g. name, address, country, email, phone, title (of paper), name (of session). The specification of name is somewhat different, and country and address is only indirectly linked to the person in our case, through the affiliation relationship, and not directly as in the IFIP-model.
- There are entity-classes that are given synonymous labels. (participant/potential participant).
- There are entity-classes in the old diagram that are represented as relationship-classes in the new diagram, e.g. review.
- There are similar entity-classes that are represented with different labels e.g. CRC vs accepted paper.
- There are similar relationship-classes that are represented with different labels, partly because the old model use role-names on relationship-classes and the new use relationship-names e.g. Presentation vs grouped_in, author vs write.

- There are aspects that are represented as sub-classes in the old version (accepted/rejected paper) which are represented as values of an attribute in the new (state.name).
- There is a type in the old diagram that is represented as a scenario in the new diagram, including values of attributes. (evaluation vs. dimension/value/scale.) The large difference here is caused by different requirements to flexibility.
- There are attributes in the old diagram that are represented as entity-classes in the new e.g. comment.
- There are similar attributes with different labels e.g. number vs reference for paper.
- There are types in the old diagram that are represented as an entity-class and a relationship-class in the new diagram (Paper cover subject vs paper.keywords and PC-member interest subject vs referee studies).

In addition to the above differences, there are areas which are included in the old diagram that are not included in the new, and vice versa.

12.3.3 Inter-organizational Model Integration

This can be interesting in several cases. A model is developed in an external organization (e.g. a model of an application framework), and one wants to find out to what extent this model is sufficiently close to a model developed within the organization for the possible adoption of the framework. The models to be compared are not necessarily written in the same language. Another possible scenario for the need of performing inter-organizational model integration is the case where several organizations needs to coordinate parts of its data-processing, for instance when applying EDI, or when temporarily cooperating to form a virtual organization. Obviously also more permanent cooperation through e.g. mergers can provoke the need for model integration of this sort.

Based on the above we would similarly expect large differences between models in inter-organizational model integration also when they are expressed in the same language. When being expressed in different languages the problems would potentially even be larger even if the languages share the same perspective.

Although matching can be done on type [251], it is often dependant on using information in the labels of the model, since there is a large number of e.g. process-diagrams that have the same port-structures and that diagrams with different port-structures might contain similar functionality. Thus even if the use of analogy for general reuse [48] can be a powerful mechanism, one can not expect to be able to use this in all cases.

12.3.4 Outline of an Approach to Model Integration

One way of attacking this area is to use a(technical) actor model as an outset for the comparisons on a high level. Features of the actor model that can be utilized for model integration purposes are:

- Data accessible by the different actors.
- The capabilities of the actors. This might both be existing capabilities and potential capabilities. The capabilities of an actor in certain roles can also be interesting.
- The type of the items that the actors might receive and send to be able to utilize the capabilities of the actors. This includes descriptions of the interface of the actor.
- The rules that applies to the actor.
- Actual and potential supporting actors of the actor.

- Different abstractions of the actor such as ONER-models and PPMs.

We have opened up for the support of matching models written in different languages, and possibly using different perspectives. To be meaningful, there must be some potential for overlap between the models of the different languages. Some examples are the comparison between an ER-model and a ONER model, where the ONER language subsume the ER-language. This can be supported by having translation facilities between the modeling languages, and perform comparisons based on transformed models [11]. More complicated is the problem of comparing say a process and a data model. As we have illustrated, there are some connection between the data and process modeling languages in the conceptual framework, thus, these can be utilized for a preliminary matching using the same principles that are used in the technique of driving questions. Another approach to this, is the use of a common underlying model, as is done in e.g. ARIES [22].

The actor-modeling language is meant to both give a view independent of traditional modeling languages, by looking upon an actor as having certain capabilities, and investigating actor support relationship between technical actors. It also cover the use of traditional data, process, and rule-modeling. It necessitates a translation from other models to this model to be useful in extended inter-organizational reuse and matching. In addition to the use of the actor-model, it should be possible to use the translations of conceptual models into statements, as outlined in Appendix E directly.

In cases where the models to be integrated do not have a common predecessor, one should as a rule try to find the potential overlap between models and not the difference, since the difference often seem to be larger than the overlap. When having found models with certain overlap, one might apply a transformational approach to matching, thus finding how many inserts and deletes of statements that must be done to transfer one model to another, by utilizing the symmetry between inserts and deletes. For models in specific languages e.g. ONER-models, one should investigate heuristics in this search to make it possible to apply the technique more efficiently. Specific approaches for merging models that have the same predecessor have been discussed in [4, 282] and is not repeated here.

12.4 Chapter Summary

We have in this chapter given an outline of what we regard to be the main contributions of this thesis, and pointed to some of the areas for further work within the approach to CIS-support in organizations that we have outlined, with specific emphasis on how to extend the framework with model integration techniques.

Although much work is left to support all the activities of system devtenance, we have in this thesis laid the foundation of what we feel is an important direction in the support of the COIS of an organization with the use of conceptual modeling techniques.

Appendix A

Case-studies

Throughout the thesis, we have presented examples from several case-studies. The main case-study is divided into two parts: The information system support at IDT, and in particular the IS-group, and the development of a conference support system. This is the main case-study, and will be given a thorough presentation here. These cases have been used specifically to investigate the extension of actor and rule modeling, and to test out parts of the suggested methodology. In addition, we have presented examples from case-studies being originally performed in the Tempora-project. Since much of the experience on the use of rules and rule-hierarchies, and the link between rules and process and data-models has come from these cases, we have also included these. Only an overall case description of these cases, the Sweden Post and the library case respectively is given here.

A.1 CIS-support at IDT

This case takes as the outset the complete portfolio of the part of the institute that we are connected to, and illustrates the link between the portfolio and the application system level by modeling in more detail a system that has been developed to help in the organization of an IFIP working conference. The organization of a conference is a widely used case, we are thus able to compare the models being prepared in connection with the development of this system using the modeling language in our conceptual framework, and other attempts of modeling this area.

It should be noted that the portfolio of a computer science institute of this kind is somewhat different from what is found in most of the organization of our survey.

- Most of the users have a very high degree of computer-literacy.
- People can temporally use many different CISs, just to test them out, i.e. not to use them on a daily basis. It is also many of the CISs that are used by very few people.
- Much of the software that is developed, is developed for the use of a small group, often only one person, to act as a validation of scientific ideas.

On the other hand, many similarities also exist, especially on the need for administrative software used by more or less computer illiterate[1] persons such as secretaries and old professors.

[1] In the sense that they have no computer science education.

A.1.1 IDT-case

The following case contains different actor-models based on the current situation of our institute. All individuals in the case-study are partly anonymized by using their initials only. The description indicate the situation as of January 1995.

An overall organizational model of the institute is given in Figure A.1. The top internal actor is IDT, the institute. As we see from the figure, this is part of a faculty (which also have additional parts), which again is one of several faculties at NTH. The institute have 11 main group, shown in the bottom of the figure, anyone being employed at the institute being part of one. In addition is there several administrational groups and committees. Also one role is specified here, being institutionalized by IDT. The same numbering is used throughout the case-studies on actors and roles in the textual description of the models.

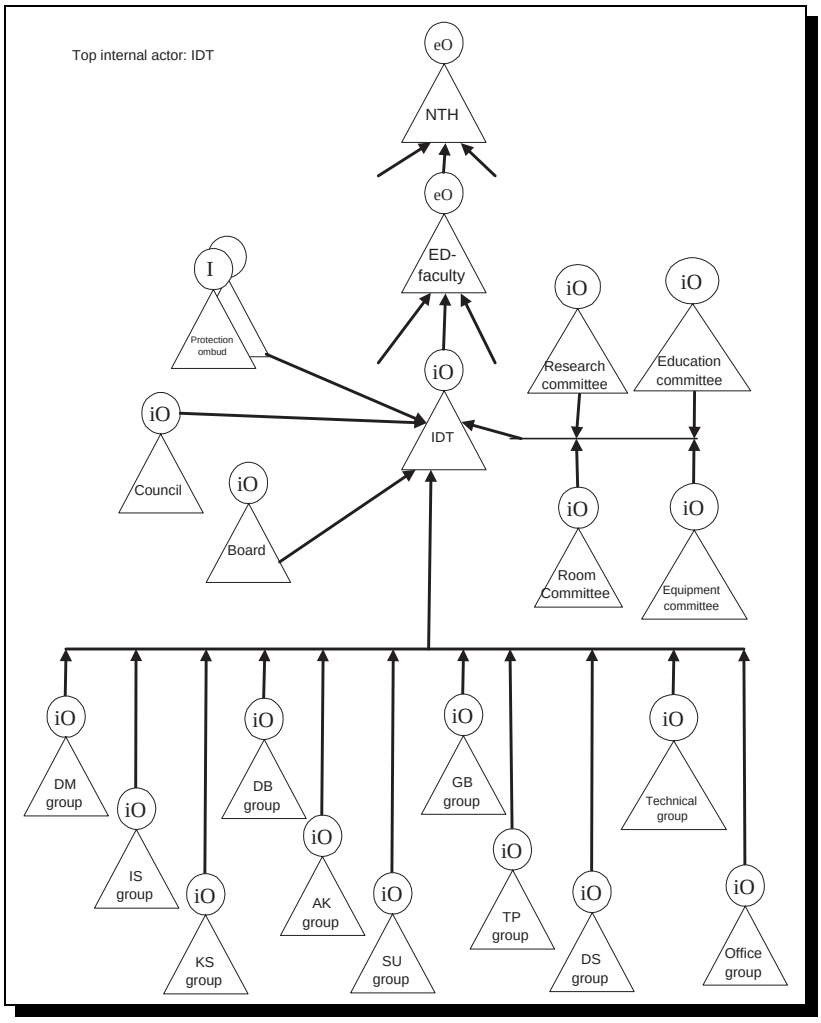

Figure A.1: Actor model describing IDT

Actors

- A1 NTH: External organizational. The Norwegian institute of technology
- A2 ED-faculty: External organizational. The faculty of electronics and computer science being one of several faculties at NTH.

- A3 IDT: Internal organizational. The institute for computer systems and telematics, being one of several institutes at the ED-faculty.
- A4 DM-group: Internal organizational. The computer hardware group at IDT.
- A5 IS-group: Internal organizational. The information systems group at IDT.
- A6 KS-group: Internal organizational. The knowledge systems group at IDT.
- A7 DB-group: Internal organizational. The database group at IDT.
- A8 AK-group: Internal organizational. The algorithm group at IDT.
- A9 SU-group: Internal organizational. The software engineering group at IDT.
- A10 GB-group: Internal organizational. The graphics and image processing group at IDT.
- A11 TP-group: Internal organizational. The network group at IDT.
- A12 DS-group: Internal organizational. The distributed systems group at IDT.
- A13 Technical group: Internal organizational. The technical support group at IDT.
- A14 Office group: Internal organizational. The secretary group at IDT.
- A15 Board: Internal organizational. The board of IDT.
- A16 Council: Internal organizational. The council of IDT.
- A17 Research committee: Internal organizational. A committee working with research oriented matter that are applicable for the whole IDT.
- A18 Education committee : Internal organizational. A committee working with overall matters regarding the education given by IDT.
- A19 Room committee : Internal organizational. A committee planning the use of the rooms that are delegated to IDT.
- A20 Equipment committee : Internal organizational. A committee deciding upon the procurement of hardware and supportive software.

Roles

- R1 Protection ombud: Has responsibility for the working environment of the institute.

Relationship	From	Role	Behalf	To	Comment
Part-of	A4			A3	
Part-of	A5			A3	
Part-of	A6			A3	
Part-of	A7			A3	
Part-of	A8			A3	
Part-of	A9			A3	
Part-of	A10			A3	
Part-of	A11			A3	
Part-of	A12			A3	
Part-of	A13			A3	
Part-of	A14			A3	
Part-of	A15			A3	
Part-of	A16			A3	
Part-of	A17			A3	
Part-of	A18			A3	
Part-of	A19			A3	
Part-of	A20			A3	
Part-of	A3			A2	
Part-of	A2			A1	
Instituted-by		R1		A3	

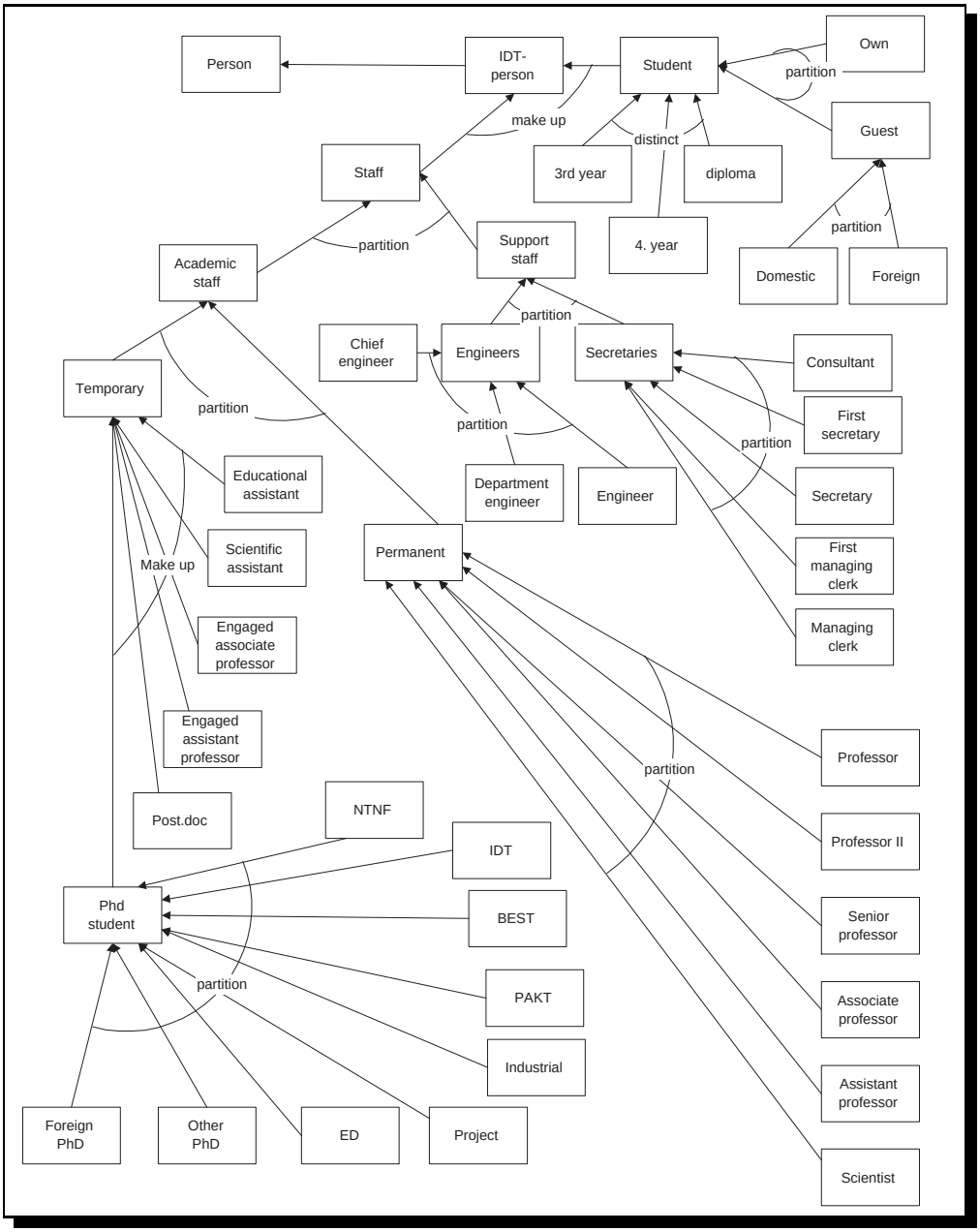

Figure A.2: ONER-model describing the static aspects of formal roles at IDT

Additional roles connected to the institute could be modeled in a role-hierarchy, and linking these roles up to a set of entities. Since the ONER-language have additional constructs for indicating the coverage of sub-classes, we have illustrated the relevant roles at the institute in an ONER-model as illustrated in Figure A.2.

Entity classes

- E1 Person.
- E2 IDT-person: Superclass E1.
- E3 Student: Engineering students at IDT. Superclass E2.
- E4 Own: Own students. Superclass E3.
- E5 Guest: Guest students. Guest students are usually diploma students (E8). Superclass E3.
- E6 3rd year: 3rd year students. Superclass E3.
- E7 4th year: 4th year students. Superclass E3.
- E8 Diploma: Diploma students. Superclass E3. See also R126.
- E9 Domestic: Domestic guest students. Superclass E5.
- E10 Foreign: Foreign guest students. Superclass E5.
- E11 Staff: People being employed by IDT. Superclass E2.
- E12 Academic staff: Teachers and researchers. Superclass E11.
- E13 Temporary: Persons being employed at IDT for a limited period. Superclass E12.
- E14 PhD student: Persons having the pursuit of a PhD-degree as their main reason for being at IDT. Superclass E13. See also R109. The further split-up of PhD-students is based on their financing.
- E15 Foreign PhD: A non-Norwegian with a foreign scholarship pursuing a PhD degree at IDT. Superclass E14.
- E16 Other PhD: A PhD-student with different financing than any of the other groups mentioned. Superclass E14.
- E17 ED: A PhD-student with a scholarship from the ED-faculty. Superclass E14.
- E18 Project: A PhD-student financed through a specific project. Superclass E14. See also R121.
- E19 Industrial: A PhD-student financed by industry. Superclass E14.
- E20 PAKT: A PhD-student being part of the PAKT-program. Superclass E14.
- E21 BEST: A PhD-student being part of the BEST-program. Superclass E14. See also R122.
- E22 IDT: A PhD-student with scholarship from IDT. Superclass E14.
- E23 NTNF: A PhD-student with scholarship from NTNF (now NFR). Superclass E14. See also R124.
- E24 Post.doc: A person with a post doctoral scholarship. Superclass E13.
- E25 Engaged assistant professor: Superclass E13.
- E26 Engaged associate professor: Superclass E13. See also R120.
- E27 Scientific assistant: Often diploma students. Superclass E13. See also R123.
- E28 Educational assistant: An educational assistant is usually a student. Superclass E13.
- E29 Permanent: A person permanently employed at IDT. Superclass E12.
- E30 Scientist: Superclass E29. See also R105.
- E31 Assistant professor: Teacher without a PhD-degree. Superclass E29.
- E32 Associate professor: Teacher with PhD-degree. Superclass E29. See also R119.
- E33 Senior professor: Specific position for old professor, having a smaller teaching-load than an ordinary professor. Superclass E29.

- E34 Professor II: Position at the professor-level being 1/5 of a full position. Superclass E29. See also R118.
- E35 Professor: Superclass E29. See also R102.
- E36 Support staff: Administrational and technical staff at the institute. Superclass E11.
- E37 Engineers: Superclass E36. See also R134.
- E38 Chief engineer: Superclass E38. See also R129.
- E39 Department engineer: Superclass E39. See also R131.
- E40 Engineer: Superclass E40.
- E41 Secretaries: Superclass E36.
- E42 Consultant: Superclass E41.
- E43 First secretary: Superclass E41.
- E44 Secretary: Superclass E41.
- E45 First managing clerk: Superclass E41. See also R127.
- E46 Managing clerk: Superclass E41. See also R128.

For all these roles/classes, one can further specify attributes and relationships. We do not present a more detailed ONER-model of the institute here. Note than when depicting the above situation in a role model using part-of relationships, the arrows would typically have been pointed in the other direction.

A role-diagram indicating both formal and informal roles, role institutionalizing, and role membership, is shown in Figure A.3, where different aspects of the role 'professor' are depicted. The role is specialized in IDT and NTH-professor, and has three subroles, scientist, lecturer, and advisor.

Actors

- A1 NTH: External organizational.
- A3 IDT: Internal organizational.
- A101 AS: Internal individual.

Roles

- R102 IDT professor: Internal individual.
- R102 NTH professor: External individual.
- R104 Professor: Individual.
- R105 Scientist: Individual.
- R106 Lecturer: Individual.
- R107 Advisor: Individual. Advisor for PhD and Diploma students.

Relationship	From	Role	Behalf	To	Comment
Part-of		R103		R102	
Part-of		R104		R102	
Part-of		R104		R103	
Part-of		R105		R104	
Part-of		R106		R104	
Part-of		R107		R104	
Fill	A101			R102	
Instituted by		R102		A3	
Instituted by		R103		A1	

Figure A.4 indicate some of the roles that an individual JK currently fill, both within and outside the institute. In also indicates the expectations of him in a given role (as a man) is formed by

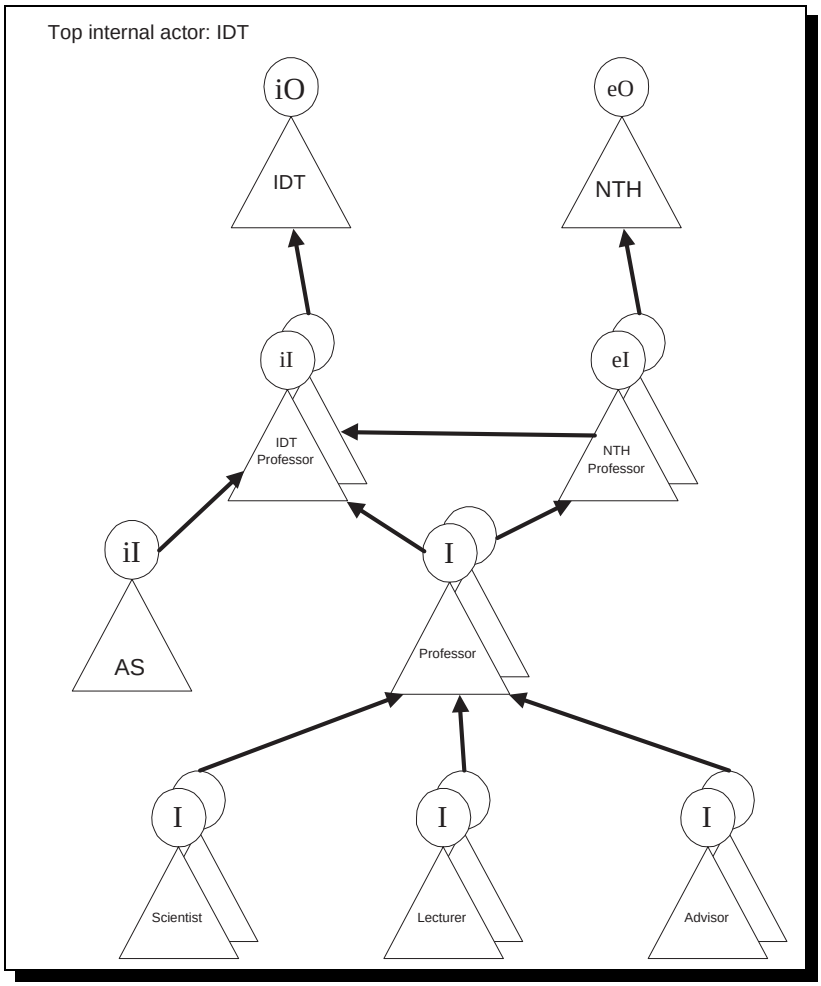

Figure A.3: Actor model describing aspects of the role 'professor'

especially BR in addition to himself. The figure also indicate how organizational actors can fill several roles.

Actors

- A1 NTH: External organizational. The Norwegian institute of technology
- A3 IDT: Internal organizational. The institute for computer systems and telematics, being one of several institutes at the ED-faculty.
- A21 ICSM95: External organizational. The organizers of the ICSM'95 conference.
- A22 ISDO95: External organizational. The organizers of the ISDO'95 conference.
- A23 AINA: External organizational. Norwegian Section of Amnesty International.
- A102 JK: Internal individual.
- A103 BR: External individual.

Roles

- R1 Workplace: Organizational.
- R2 Institute: Organizational.
- R108 PC-member: Individual. Member of a program committee of a conference.
- R109 PhD student: Internal individual.
- R110 Group consultant: External individual.

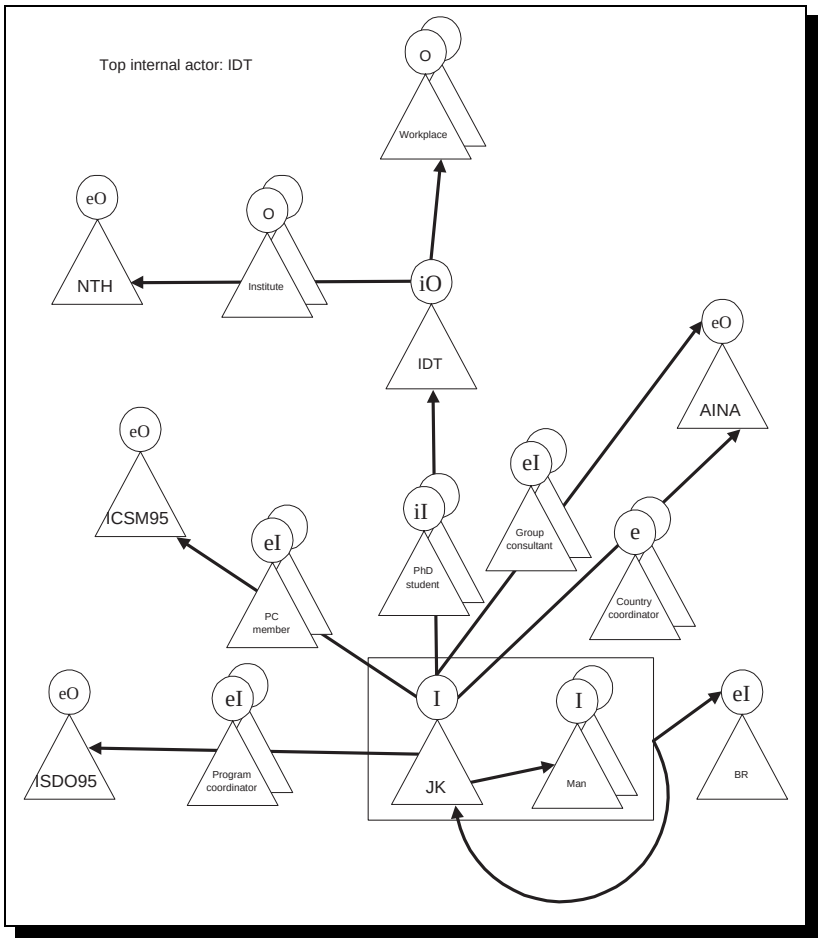

Figure A.4: Actor model describing actors with multiple simultaneous roles

- R111 Country coordinator: External.
- R112 Program coordinator: External individual. Position within the organizing committee of a professional conference.
- R113 Man: Individual.

Agents

- AG1 JK in the role 'man'.

Relationship	From	Role	Behalf	To	Comment
Part-of	A3	R2		A1	
Part-of	A102	R109		A3	
Part-of	A102	R112		A22	
Part-of	A102	R108		A21	
Part-of	A102	R110		A23	
Part-of	A102	R111		A23	
Fill	A3			R1	
Instituted by	AG1			A102	
Instituted by	AG1			A103	

In Figure A.5 we have exploded one of the administrational organizational actors in Figure A.1, the education committee. The internal organizational actor here is the education committee itself.

The different members of the committee, including the role they have in the actual committee (internal roles) and also the organizational actor they represent in this committee, are indicated. The two positions reserved for IDT-students, are currently vacant. Similar diagrams can be made for the other administrative organizational actors in Figure A.1.

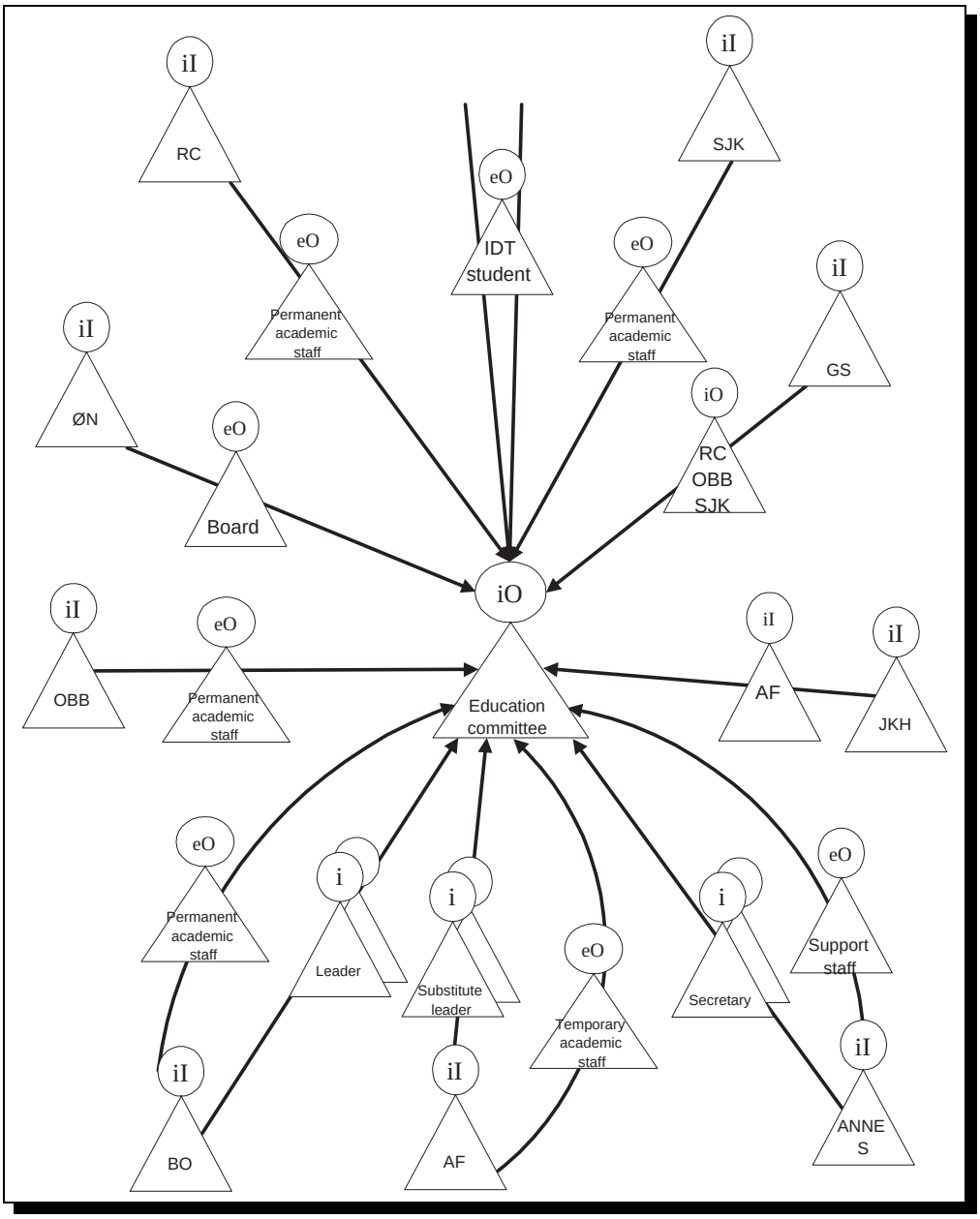

Figure A.5: Actor model describing the education committee

Actors

- A15 Board: External organizational. The board of IDT.
- A18 Education committee : Internal organizational. A committee working with overall matters regarding the education given by IDT.
- A24 Permanent academic staff: External organizational.
- A25 IDT-student: External organizational.
- A26 RC,OBB,SJK: Internal organizational. An actor consisting of these three individuals.
- A27 Temporary academic staff: External organizational.
- A47 Support staff: External organizational.
- A104 RC: Internal individual.
- A105 ØN: Internal individual.
- A106 OBB: Internal individual.
- A107 BO: Internal individual.
- A108 AF: Internal individual.
- A109 ANNES: Internal individual.
- A110 JKH: Internal individual.
- A111 GS: Internal individual.
- A112 SJK: Internal individual.

Roles

- R114 Leader: Internal individual. The leader of the education committee.
- R115 Substitute leader: Internal individual. The substitute leader of the education committee, representing the committee in other committees when the leader of the committee is not able to.
- R116 Secretary: Internal individual. The secretary of the education committee.

	From			*To*		
Name	Node	Role	Behalf	Node	Role	Behalf
Part-of	A104		A24	A18		
Part-of	A105		A15	A18		
Part-of	A106		A24	A18		
Part-of	A107	R114		A18		
Part-of	A107		A24	A18		
Part-of	A108	R115		A18		
Part-of	A108		A27	A18		
Part-of	A109	R116		A18		
Part-of	A109		A47	A18		
Part-of	A110		A108	A18		
Part-of	A111		A26	A18		
Part-of	A112		A24	A18		
Part-of	ANNN		A25	A18		
Part-of	ANNN		A25	A18		

Figure A.6 indicates the overall work-pattern of the committee: The leader, BO, receive case-material, both from the members of the educational committee and from the board, but also from actors external to IDT, such as UVUS, the similar committee at the faculty-level, and IFI, the institute for Informatics at the university level.

Based on the number and urgency of the incoming cases, a meeting agenda is set up and distributed to the members of the committee. The secretary ANNE S is responsible for this.

The meeting is held a number of days after the agenda is distributed, and start with the approval or disapproval of minutes from previous meetings. If these are approved, they are distributed to the rest of the institute.

Based on the meeting, the secretary prepares preliminary minutes, which are sent by email to the committee members present at the meeting for approval. A completed minute is prepared based on the responses on this preliminary minute, and is finally approved in the next meeting of the committee.

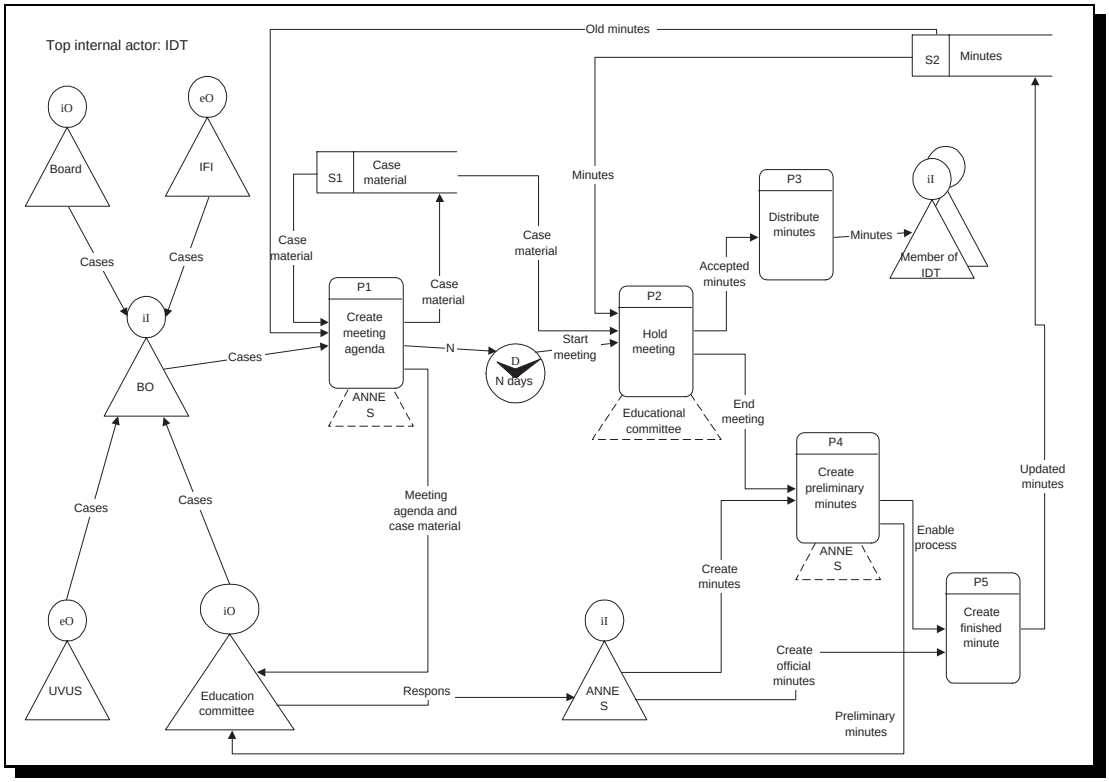

Figure A.6: PPM describing the working pattern of the education committee

Actors

- A15 Board: Internal organizational. The board of IDT.
- A18 Education committee : Internal organizational. A committee working with overall matters regarding the education given by IDT.
- A28 IFI: External organizational. Institute for Informatics at the University of Trondheim.
- A29 UVUS: External organizational. A committee working with overall matters regarding the education given by the ED-faculty.
- A107 BO: Internal individual.
- A109 ANNES: Internal individual.

Roles

- R117 Member of IDT.

Stores

- S1 Case material: Background material for the work in the committee.
- S2 Minutes: Previous official minutes from meetings being held in the committee.

Timers

- D N days: The number of days between the day the meeting agenda is created and the actual meeting takes place. Used to indicate the temporal relationship between these two processes.

Processes

- P1 Create meeting agenda.
- P2 Hold meeting.
- P3 Distribute minutes.
- P4 Create preliminary minutes: Create a first draft of the minutes, which are made available by email to the committee.
- P5 Create finished minutes.

Flows

- f1 Cases: Suggestions for cases to be discussed in the education committee.
- f2 Cases.
- f3 Cases.
- f4 Cases.
- f5 Cases.
- f6 Old minutes: Minutes from previous meetings.
- f7 Case material.
- f8 N: Number of days to next meeting.
- f9 Case material.
- f10 Meeting agenda and case material.
- f11 Start meeting: Signal.
- f12 Case material.
- f13 Minutes.
- f14 Accepted minutes.
- f15 End meeting: Signal.
- f16 Create minutes: Signal.
- f17 Enable process: Signal.
- f18 Preliminary minutes.
- f19 Create official minutes.
- f20 Updated minutes.
- f21 Minutes.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	A15			A107		
f2	A28			A107		
f3	A29			A107		
f4	A18			A107		
f5	A197			P1		
f6	S2			P1		
f7	S1			P1		
f8	P1			D		
f9	P1			S1		
f10	P1			A18		
f11	D			P2		

continued on next page

continued from previous page						
	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f12	S1			P2		
f13	S2			P2		
f14	P2			P3		
f15	P2			P4		
f16	A109			P4		
f17	P4			P5		
f18	P4			A18		
f19	A109			P5		
f20	P5			S2		
f21	P3				R117	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A18			P2				
A109			P1				
A109			P4				

In Figure A.7, an overview is given of the IS-group. The internal actor is the group, and the internal and external roles the members fill in the group and at IDT respectively are illustrated. When a role is positioned over several member-of relations, this means that the actors have the same role within the organizational actor. This is one example of expressive economy of actor-diagrams. Similar figures can be made for the other groups, but are not indicated here.

The figure also contains an overview of persons supporting members of the IS-group at IDT, being either secretaries or engineers.

Actors

- A2 ED-faculty: External organizational. The faculty of electronics and computer science.
- A3 IDT: External organizational. The institute for computer systems and telematics.
- A5 IS-group: Internal organizational. The information systems group at IDT.
- A13 Technical group: External organizational. The technical support group at IDT.
- A14 Office group: External organizational. The secretary group at IDT.
- A16 Council: External organizational. The council of IDT.
- A17 Research committee: External organizational. A committee working with research oriented matter that are applicable for the whole IDT.
- A101 AS: Internal individual.
- A102 JK: Internal individual.
- A113 PH: Internal individual.
- A114 RA: Internal individual.
- A115 OIL: Internal individual.
- A116 AHS: Internal individual.
- A117 GB: Internal individual.
- A118 HR: Internal individual.
- A119 MY: Internal individual.
- A120 BAF: Internal individual.

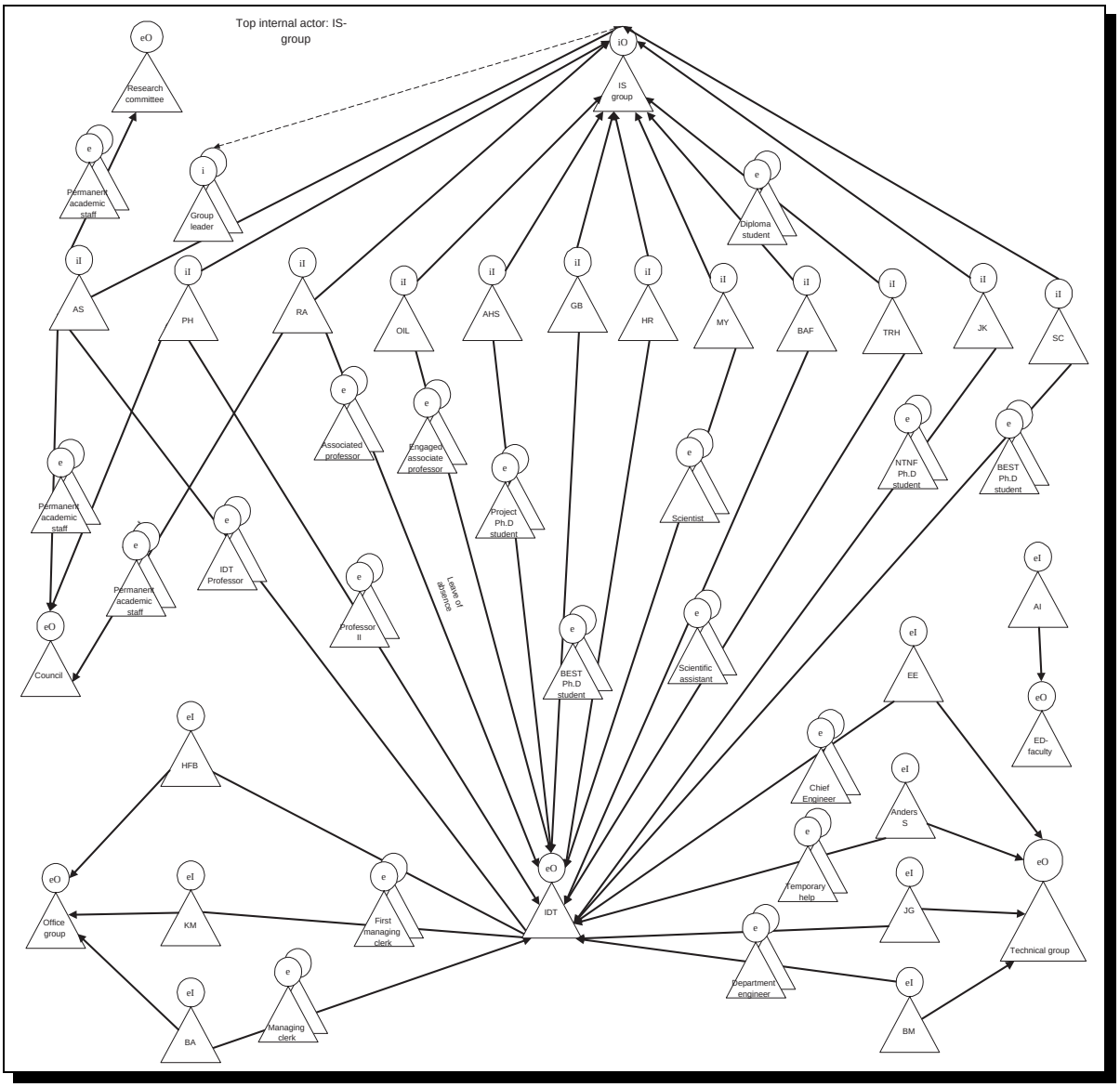

Figure A.7: Actor model describing the IS group

- A121 TRH: Internal individual.
- A122 SC: Internal individual.
- A123 HFB: External individual.
- A124 KM: External individual.
- A125 BA: External individual.
- A126 EE: External individual.
- A127 AndersS: External individual.
- A128 JG: External individual.
- A129 BM: External individual.
- A130 AI: External individual.

Roles: For further description of the roles, see Figure A.2.

- R103 IDT professor: External individual.
- R105 Scientist: External individual.

- R118 Professor II: External individual.
- R119 Associate professor: External individual. Currently on leave of absence.
- R120 Engaged associate professor: External individual.
- R121 Project PhD student: External individual.
- R122 Best PhD student: External individual.
- R123 Scientific assistant: External individual.
- R124 NTNF PhD student: External individual.
- R125 Diploma student: External individual.
- R127 First managing clerk: External individual.
- R128 Managing clerk: External individual.
- R129 Chief engineer: External individual.
- R130 Temporary help: External individual.
- R131 Department engineer: External individual.
- R132 Permanent academic staff: External organizational.
- R133 Group leader: Internal individual.

Relationship	From	Role	Behalf	To	Comment
Part-of	A101	R103		A3	
Part-of	A102	R124		A3	
Part-of	A113	R118		A3	
Part-of	A114	R119		A3	Leave of absence
Part-of	A115	R120		A3	
Part-of	A116	R121		A3	
Part-of	A117	R122		A3	
Part-of	A118	R122		A3	
Part-of	A119	R105		A3	
Part-of	A120	R123		A3	
Part-of	A121	R123		A3	
Part-of	A122	R122		A3	
Part-of	A123	R127		A3	
Part-of	A124	R127		A3	
Part-of	A125	R128		A3	
Part-of	A126	R129		A3	
Part-of	A127	R130		A3	
Part-of	A128	R131		A3	
Part-of	A129	R131		A3	
Part-of	A101	R133		A5	
Part-of	A102			A5	
Part-of	A113			A5	
Part-of	A114			A5	
Part-of	A115			A5	
Part-of	A116			A5	
Part-of	A117			A5	
Part-of	A118			A5	
Part-of	A119			A5	
Part-of	A120	R126		A5	

continued on next page

continued from previous page					
Relationship	From	Role	Behalf	To	Comment
Part-of	A121	R126		A5	
Part-of	A122			A5	
Part-of	A123			A14	
Part-of	A124			A14	
Part-of	A125			A14	
Part-of	A126			A13	
Part-of	A127			A13	
Part-of	A128			A13	
Part-of	A129			A13	
Part-of	A101	R132		A17	
Part-of	A101	R132		A16	
Part-of	A113	R132		A16	
Part-of	A114	R132		A16	Leave of absence
Part-of	A130			A2	
Power	A5			R133	

The rest of the data being presented for this part of the case are based on the results from a survey investigation being performed among the members of the IS-group and the secretaries being linked professionally to members of the group. Results from the investigation were confirmed with the respondents afterwards.

In Figure A.8 we have illustrated how the group-members are supported by both the office-group and the technical group. The top internal actor is the IS-group. The structure of the technical group is also illustrated. We see that e.g. JK is supported both by Unix-support generally, and one of its members, JG specifically, he in the role of engineer. JK is supported by these in two roles, both as a Ph.D. student, and in the role of program coordinator of a conference.

Similar relationships are given between the other members of the IS-group and the technical group.

When it comes to the secretaries, we see that all of them support the same set of individual actors, some also being outside the group.

Actors

- A13 Technical group: External organizational. The technical support group at IDT.
- A14 Office group: External organizational. The secretary group at IDT.
- A30 Unix-support: External organizational.
- A31 PC-support: External organizational.
- A101 AS: Internal individual.
- A102 JK: Internal individual.
- A114 RC: External individual.
- A113 PH: Internal individual.
- A114 RA: Internal individual.
- A115 OIL: Internal individual.
- A116 AHS: Internal individual.
- A117 GB: Internal individual.
- A118 HR: Internal individual.
- A119 MY: Internal individual.
- A120 BAF: Internal individual.

- A121 TRH: Internal individual.
- A122 SC: Internal individual.
- A123 HFB: External individual.
- A124 KM: External individual.
- A125 BA: External individual.
- A126 EE: External individual.
- A127 AndersS: External individual.
- A128 JG: External individual.
- A129 BM: External individual.
- A130 AI: External individual.
- A131 TR: External individual.
- A132 JANK: External individual.

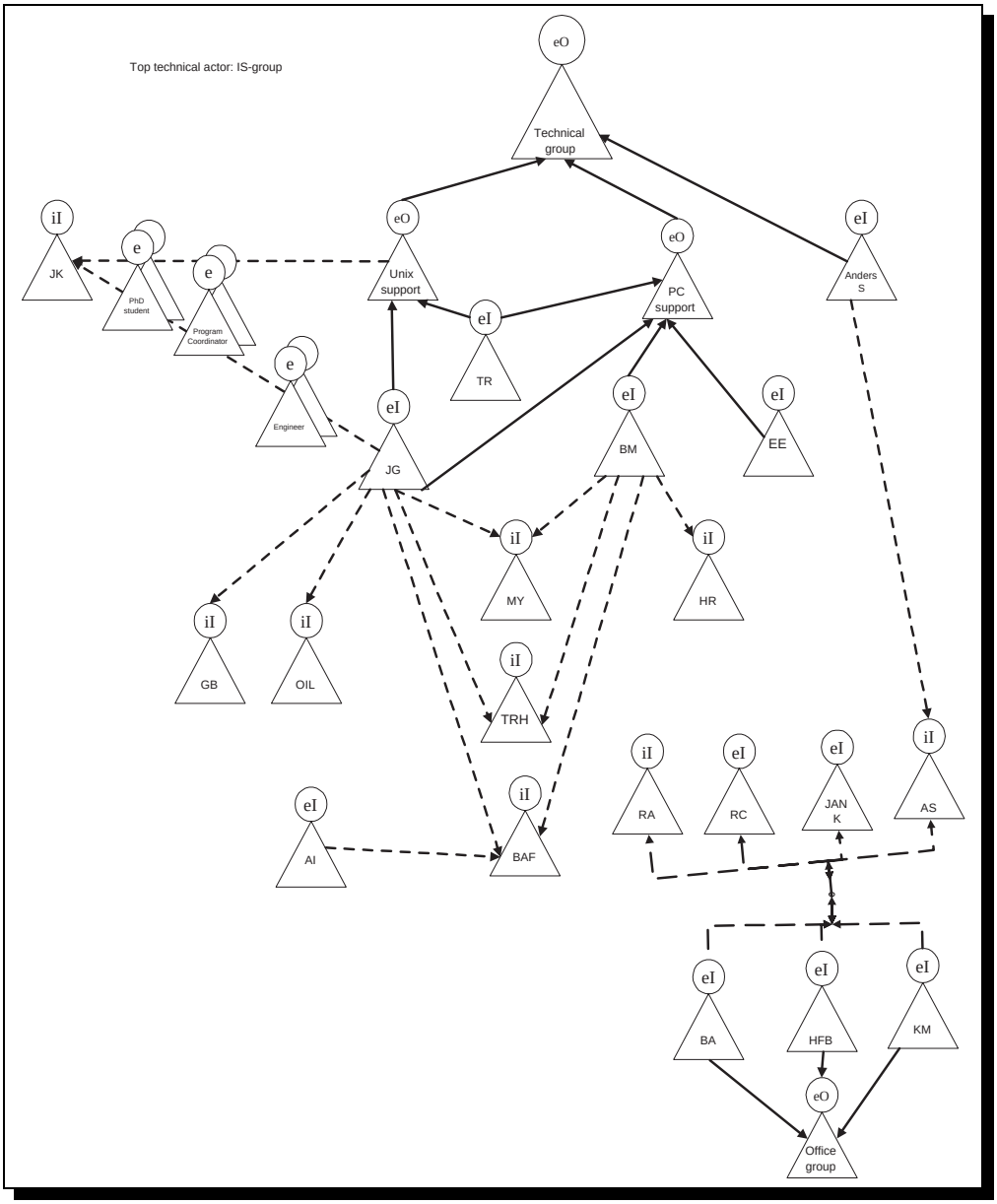

Figure A.8: Actor model describing social support relationships involving the IS-group

Roles

- R109 PhD student: External individual.
- R112 Program coordinator: External individual.
- R134 Engineer: External individual.

Relationship	From	Role	Behalf	To	Comment
Part-of	A30			A13	
Part-of	A31			A13	
Part-of	A123			A14	
Part-of	A124			A14	
Part-of	A125			A14	
Part-of	A126			A31	
Part-of	A127			A13	
Part-of	A128			A30	
Part-of	A128			A31	
Part-of	A129			A31	
Part-of	A131			A30	
Part-of	A131			A31	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A30			A102	R109			
A30			A102	R112			
A123			A101				
A123			A104				
A123			A114				
A123			A132				
A124			A101				
A124			A104				
A124			A114				
A124			A132				
A125			A101				
A125			A104				
A125			A114				
A125			A132				
A127			A101				
A128	R134		A102	R109			
A128	R134		A102	R112			
A128			A115				
A128			A117				
A128			A119				
A128			A120				
A128			A121				
A129			A118				
A129			A119				

continued on next page

continued from previous page

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A129			A120				
A129			A121				
A130			A120				

Portfolio

The CISs used by the members of the IS-group and the related secretaries are given in the below tables. The first table shows the systems used, what platforms they run on and if they are supportive or applicative. For both kinds is indicated the external supportive actors (e.g. vendors of the supportive actors). In addition it illustrates the related secretaries that use them. Supportive actors are divided into developer supportive, office supportive, and general supportive as defined in Appendix H.

Tool	S/A	Vendor	PC	Unix	MAC	HFB	KM	BA
PC						X	X	X
Unix								
MAC								
Windows/DOS	GS	Microsoft	X			X	X	X
Norton Desktop	GS	Norton	X			X	X	X
Word	OS	Microsoft	X			X	X	X
Word Perfect	OS	Word Perfect	X		X	X		X
Amipro	OS	Lotus	X					
Lotus 123	OS	Lotus	X				X	X
Excel	OS	Microsoft	X				X	X
Powerpoint	OS	Microsoft	X					
Access	DS	Microsoft	X					
Borland C++	DS	Borland	X					
Zapp C++	DS	zAPP	X					
NFS-mail	OS		X			X	X	X
PC Eudora	OS		X					
Selectmail	OS	Sun	X					
Designer	DS		X					
Clue	OS		X					
SPSS	OS	SPSS	X					
Visio	DS	Shapeware	X					
DIP DB	A	pc-drift	X			X		
Brevjournal	A	pc-drift	X			X	X	X
Adrreg	A	pc-drift	X			X	X	X
Solaris/Sunos	GS	Sun		X				
X11	GS	MIT		X				
Motif	GS	MIT		X				
framemaker	OS	Frame	X	X				

continued on next page

continued from previous page

Tool	S/A	Vendor	PC	Unix	MAC	HFB	KM	BA
emacs	OS	GNU	X	X				
latex	OS	GNU	X	X				
bibtex	OS	GNU		X				
xdvi	OS	GNU	X	X				
dvips	OS	Radical Eye		X				
ghostview	OS	GNU		X				
idraw	OS	Stanford			X			
ftp	OS	SUN	X	X				
telix	OS	Connect	X					
kermit	OS		X	X				
www	GS	CERN		X				
mosaic	OS	NCSA	X	X				
Netscape	OS	Netscape	X	X				
lynx	OS			X				X
latex2html	OS			X				
rtftohtml	OS	Cray		X				
Unix mail	OS	Sun		X				
Z-mail	OS	Z-code		X				
X-mail	OS			X				
mailtool	OS	Sun		X				
xmh	OS	MIT		X				
xrn	OS	MIT		X				
trn	OS			X				
Sybase DB	DS	Sybase	X	X				
ingres DB	DS	Ingres		X				
Ramatic	DS	SISU		X				
Bourne shell	GS	Sun		X				
awk	DS	GNU		X				
make	DS	GNU		X				
RCS	DS	GNU		X				
gcc	DS	GNU		X				
SP	A	PH		X				
C4	A	JK		X				
bibutil	A	JK		X				
filestruc	A	JK		X				
webcreate	A	JK		X				
awho	A	BM		X				
tlf	A	Unix-drift		X				

The next table illustrate the direct users of the portfolio by the academic staff of the group.

Tool	BAF	TRH	MY	AS	HR	OIL	AHS	RA	GB	JK
PC	X	X	X	X	X	X		X		X
Unix	X	X	X			X	X		X	X

continued on next page

continued from previous page

Tool	BAF	TRH	MY	AS	HR	OIL	AHS	RA	GB	JK
MAC							X	X		
Windows/DOS	X	X	X	X	X	X	X	X		X
Norton Desktop	X	X	X	X	X	X	X	X		X
Word	X	X	X	X	X					X
Word Perfect				X						
Amipro			X							
Lotus 123										
Excel										X
Powerpoint					X	X	X			
Access					X					
Borland C++	X	X	X							
Zapp C++	X	X	X							
NFS-mail										
PC Eudora					X					
Selectmail					X					
Designer					X					
Clue					X					
SPSS										X
Visio					X					X
DIP DB										
Brevjournal										
Adrreg										
Solaris/Sunos	X	X	X			X	X		X	X
X11	X	X	X			X	X		X	X
Motif										X
framemaker		X	X							
emacs	X					X	X		X	X
latex	X	X				X	X		X	X
bibtex	X	X				X	X		X	X
xdvi						X	X		X	X
dvips						X	X		X	X
ghostview						X	X		X	X
idraw		X				X	X		X	X
ftp						X	X			X
Telix										X
Kermit										X
WWW										X
mosaic				X	X	X				X
netscape	X									X
lynx										X
latex2html										X
rtftohtml										X

continued on next page

continued from previous page										
Tool	BAF	TRH	MY	AS	HR	OIL	AHS	RA	GB	JK
Unix mail			X							
Z-mail	X									
X-mail		X								
Mailtool						X				
xmh										X
xrn						X				X
trn			X							
Sybase			X							
ingres										X
Ramatic							X			X
Bourne shell										X
awk										X
make										X
RCS										X
gcc										X
SP									X	
C4										X
bibutil										X
filestruc										X
webcreate										X
awho										X
tlf										X

In Figure A.9, it is depicted graphically who is supported by and who supports a selected technical actor, in this case Word. In this figure, the internal actor is IDT.

Actors

- A31 PC-support: Internal organizational.
- A32 Microsoft: External organizational. Software developer and vendor.
- A101 AS: Internal individual.
- A102 JK: Internal individual.
- A118 HR: Internal individual.
- A119 MY: Internal individual.
- A120 BAF: Internal individual.
- A121 TRH: Internal individual.
- A123 HFB: Internal individual.
- A124 KM: Internal individual.
- A125 BA: Internal individual.
- A201 Word: Internal supportive.
- A202 Windows: Internal supportive.

Roles

- R201 Text editor: Software.

Relationship	From	Role	Behalf	To	Comment
Fills	A201			R201	

Support relationships

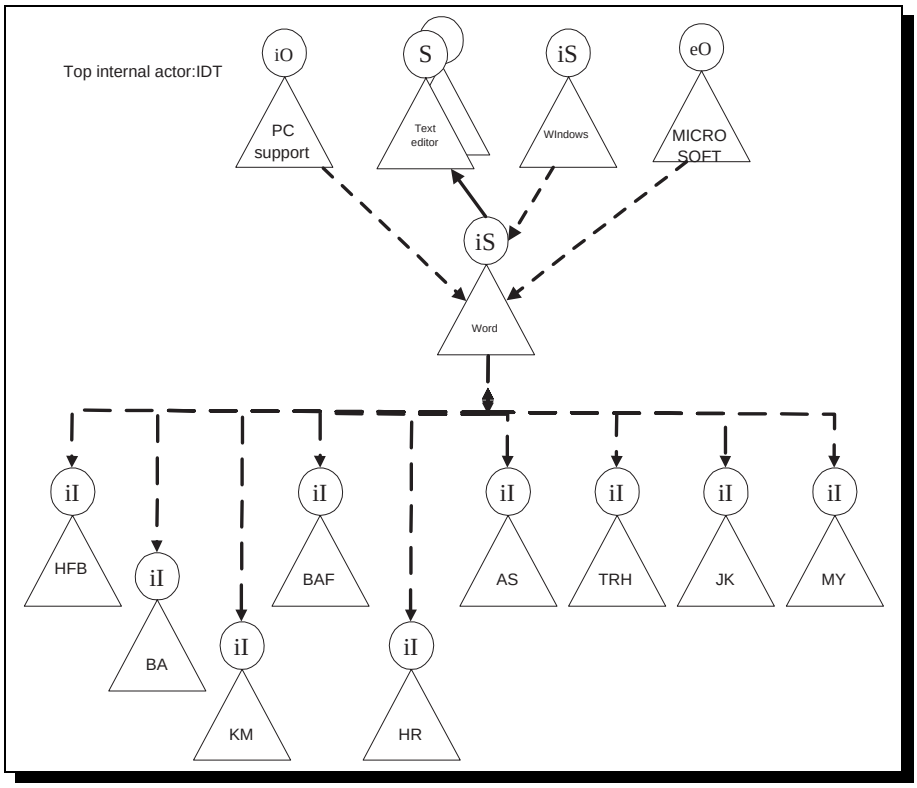

Figure A.9: Actor model describing support-relations between technical and social actors

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A31			A201				
A32			A201				
A201			A101				
A201			A102				
A201			A118				
A201			A119				
A201			A120				
A201			A121				
A201			A123				
A201			A124				
A201			A125				
A202			A201				

In Figure A.10 we have depicted graphically the set of systems that support one of the social individual actors, JK, and also the systems he support, and the additional actors being supported by these. It also include hardware actors, which as we see in the example can also be perceived to fill roles. The internal actor in the figure is IDT.

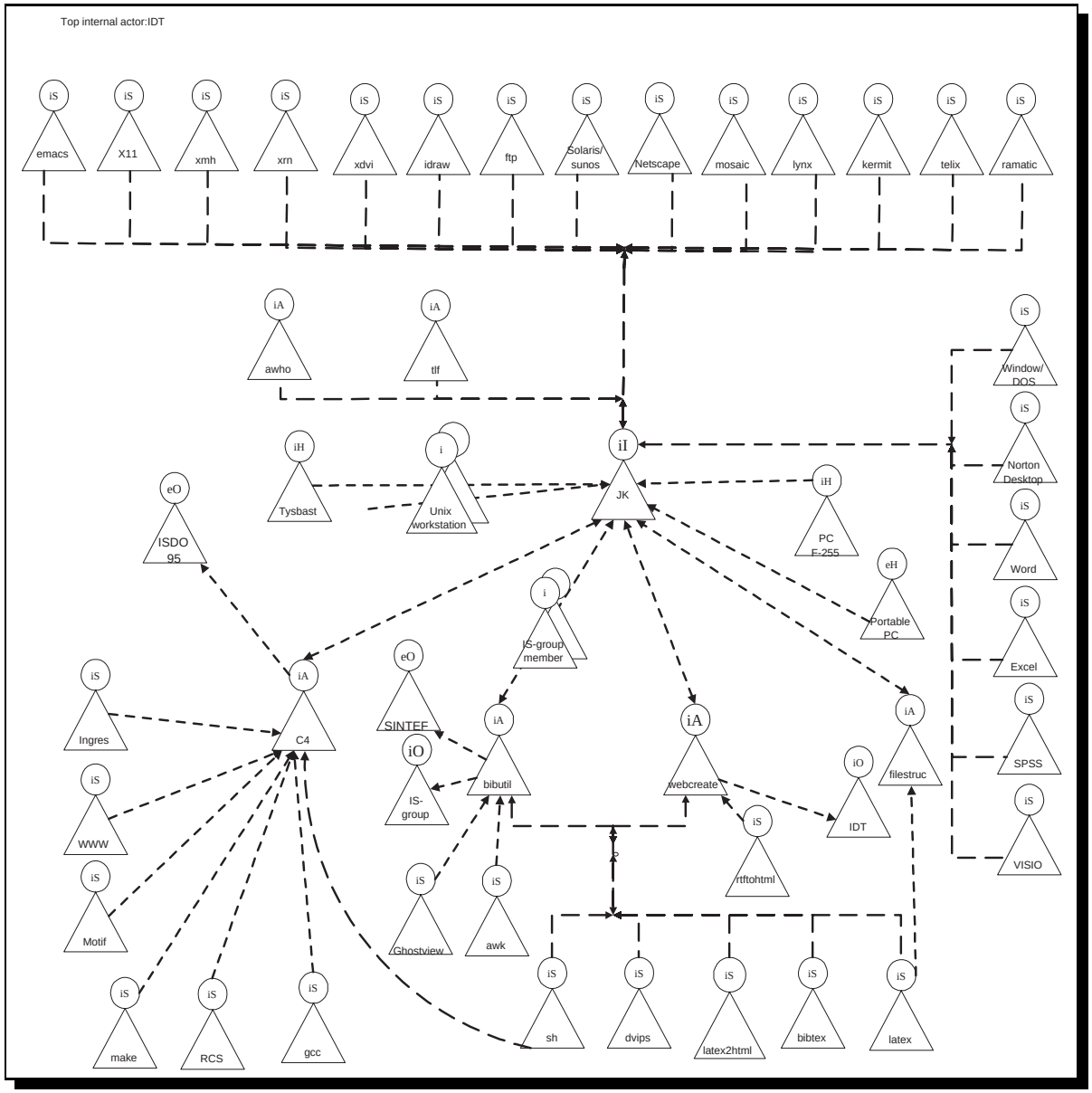

Figure A.10: Actor model describing all technical actors supporting a selected individual actor

Actors

- A3 IDT: Internal organizational. The institute for computer systems and telematics, being one of several institutes at the ED-faculty.
- A5 IS-group: Internal organizational.
- A22 ISDO95: External organizational.
- A33 SINTEF: External organizational.
- A102 JK: Internal individual.
- A201 word: Internal supportive.
- A202 Windows: Internal supportive.
- A203 emacs: Internal supportive.
- A204 X11: Internal supportive.
- A205 xmh: Internal supportive.
- A206 xrn: Internal supportive.
- A207 xdvi: Internal supportive.
- A208 idraw: Internal supportive.
- A209 ftp: Internal supportive.
- A210 Solaris/Sunos: Internal supportive. It might be better to model this as two actors.
- A211 Netscape: Internal supportive.
- A212 mosaic: Internal supportive.
- A213 lynx: Internal supportive.
- A214 kermit: Internal supportive.
- A215 telix: Internal supportive.
- A216 Ramatic: Internal supportive.
- A217 Norton desktop: Internal supportive.
- A218 Excel: Internal supportive.
- A219 SPSS: Internal supportive.
- A220 VISIO: Internal supportive.
- A221 awho: Internal applicative.
- A222 tlf: Internal applicative.
- A224 ingres: Internal supportive.
- A225 WWW : Internal supportive.
- A226 Motif: Internal supportive.
- A227 make: Internal supportive.
- A228 RCS: Internal supportive.
- A229 gcc: Internal supportive.
- A230 C4: Internal applicative.
- A231 ghostview: Internal supportive.
- A232 awk: Internal supportive.
- A233 bibutil: Internal applicative.
- A234 webcreate: Internal applicative
- A235 rtftohtml: Internal supportive.
- A236 sh: Internal supportive.
- A237 dvips: Internal supportive.
- A238 latex2html: Internal supportive.
- A239 bibtex: Internal supportive.
- A240 latex: Internal supportive.
- A241 filestruc: Internal applicative.

- A501 PC F-255: Internal hardware.
- A502 Portable PC: External hardware. PC owned by JK.
- A503 Tysbast : Internal hardware. Also currently positioned at F-255.

Roles

- R35 IS-group member. Internal individual. Related to A5.
- R501 Unix workstation: Hardware.

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A102			A230				
A102			A234				
A102			A241				
A102	R35		A253				
A201			A102				
A202			A102				
A203			A102				
A204			A102				
A205			A102				
A206			A102				
A207			A102				
A208			A102				
A209			A102				
A210			A102				
A211			A102				
A212			A102				
A213			A102				
A214			A102				
A215			A102				
A216			A102				
A217			A102				
A218			A102				
A219			A102				
A220			A102				
A221			A102				
A222			A102				
A223			A230				
A224			A230				
A225			A230				
A226			A230				
A227			A230				
A228			A230				
A229			A230				
A230			A22				
A230			A102				
A231			A233				

continued on next page

continued from previous page

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A232			A233				
A233			A102				
A233			A33				
A234			A3				
A234			A102				
A235			A234				
A236			A230				
A236			A233				
A236			A234				
A237			A233				
A237			A234				
A238			A233				
A238			A234				
A239			A233				
A239			A234				
A240			A233				
A240			A234				
A240			A241				
A241			A102				
A501			A102				
A502			A102				
A503	R501		A102				

In Figure A.11, it is indicated how the software-role data-store, can be specialized and how the specific databases position itself within this categorization. The top internal actor in the figure is IDT, thus the we can see which systems that are is available at IDT.

Actors

- A202 Windows/DOS: Internal software.
- A210 Solaris/Sunos: Internal software.
- A224 Ingres: Internal software.
- A242 Access: Internal software.
- A243 Oracle: Software.
- A244 Sybase: Internal software.

Roles

- R202 Data store: Software.
- R203 DBMS: Software.
- R204 Operating system: Software.
- R205 Unix OS: Software.
- R206 PC OS: Software.
- R207 Codasyl DB: Software.
- R208 Network DB: Software.
- R209 Relational DB: Software.
- R210 OODB: Software.

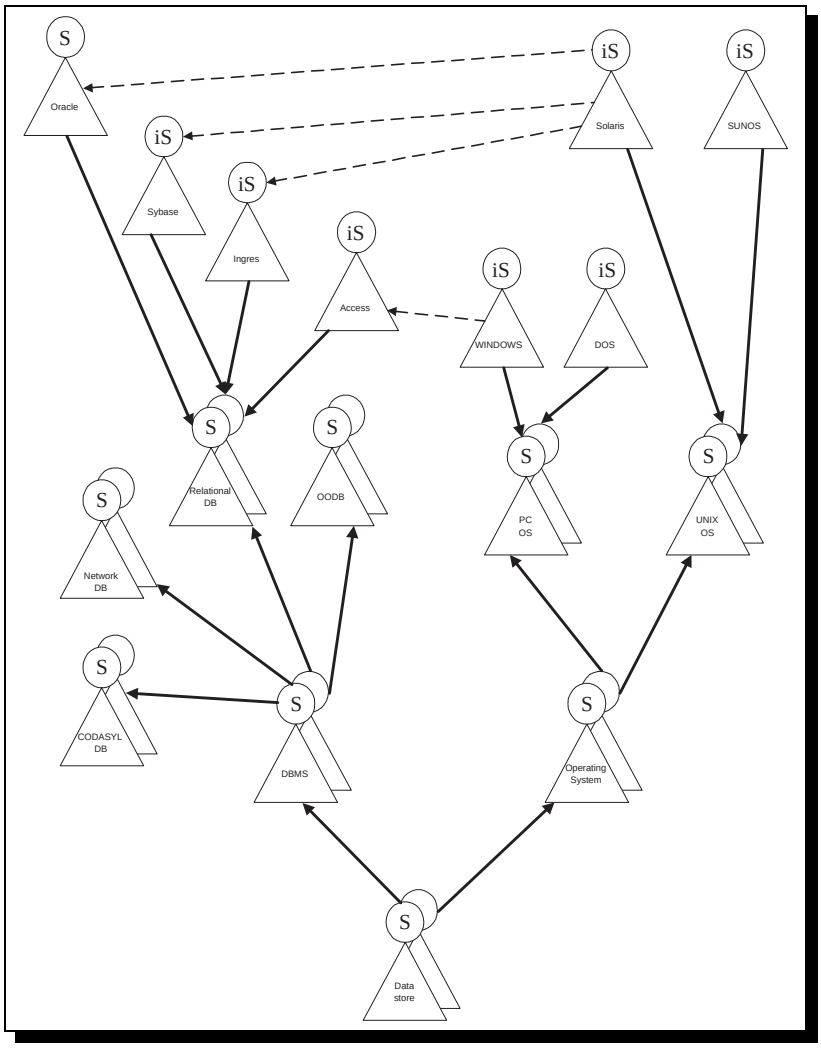

Figure A.11: Actor model describing roles of data-stores

Relationship	From	Role	Behalf	To	Comment
Fills	A202			R206	
Fills	A210			R205	
Fills	A224			R209	
Fills	A242			R209	
Fills	A243			R209	
Fills	A244			R209	
Part-of		R202		R203	
Part-of		R202		R204	
Part-of		R203		R207	
Part-of		R203		R208	
Part-of		R203		R209	
Part-of		R203		R210	
Part-of		R204		R205	
Part-of		R204		R206	

The functionality of a DBMS on the instance level as described in [254] can be depicted as in Figure A.12. Note that the DBMS-role symbol is in a sense superfluous, but is added to make the figure self-contained. No further description of this model is given.

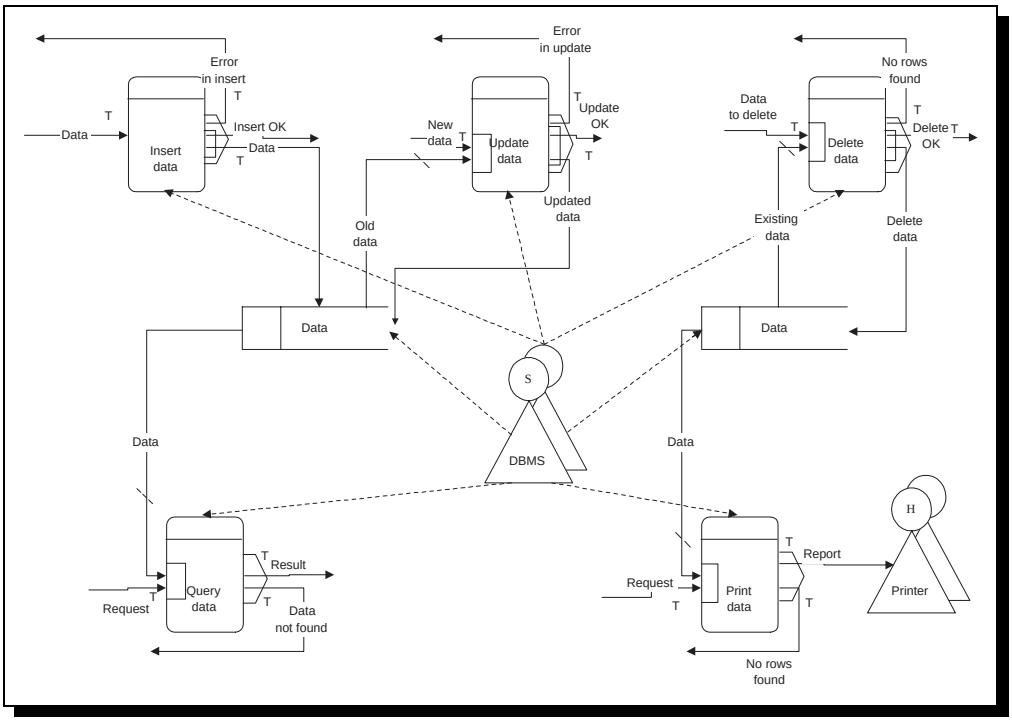

Figure A.12: PPM describing basic capabilities of a DBMS

When specializing for different kinds of DBMSs, one would need to specialize the functionality. This is not indicated here.

To further illustrate interaction between technical actors, in Figure A.13 we first show the situation where SPSS is used to create postscript-figures from statistical data to be included in a latex file on Unix. In this case, all steps are initiated by an individual social actor, in this case JK, and the arrows shows the interaction.

Actors

- A102 JK: Internal individual.
- A201 Word: Internal supportive.
- A202 DOS: Internal supportive.
- A203 emacs: Internal supportive.
- A209 ftp: Internal supportive.
- A210 Sunos: Internal supportive.
- A218 Excel: Internal supportive.
- A219 SPSS: Internal supportive.
- A237 dvips: Internal supportive.
- A239 bibtex: Internal supportive.
- A240 latex: Internal supportive.

Flows

- f1: JK enters statistical data into Excel.
- f2: The statistical data is stored in a spreadsheet format in the file system.

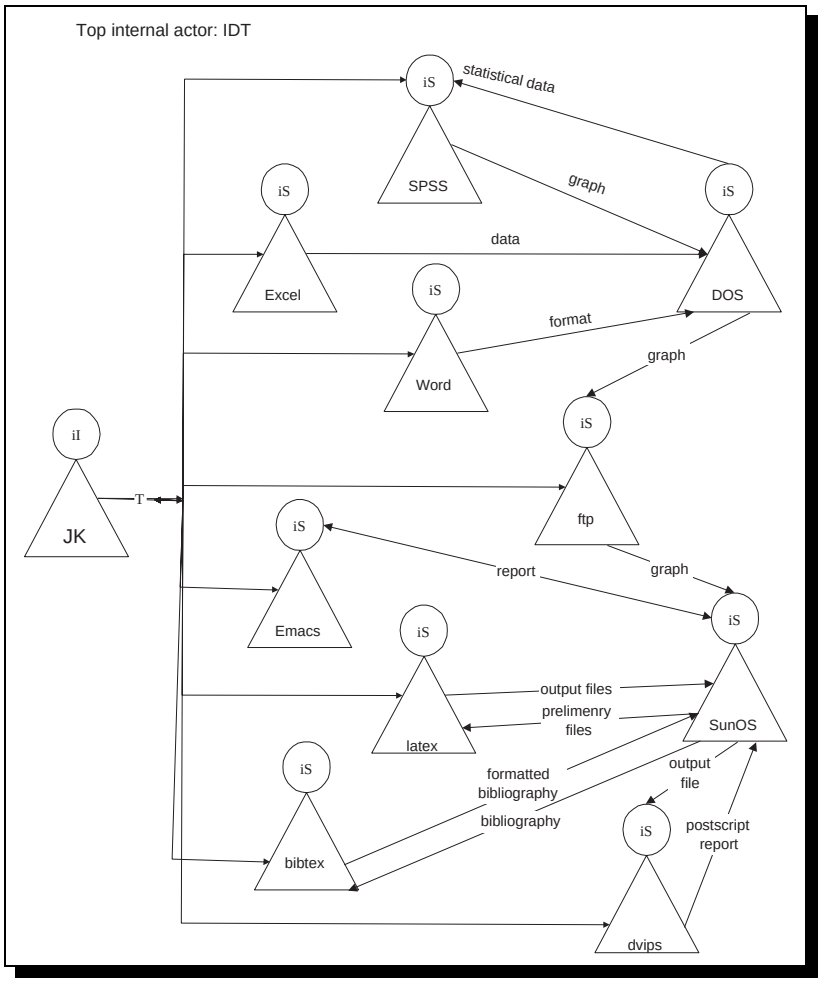

Figure A.13: Actor model describing a communication pattern between social and technical actors

- f3: JK enter setup-information to be used with the statistical data.
- f4: The set-up is stored in a plain text format in the file system.
- f5: JK uses SPSS.
- f6 statistical data: The set-up file and the statistical data in the spreadsheet file is taken into SPSS.
- f7 graph: Based on statistical analysis, a graph is created which are saved in the file system as a postscript file.
- f8: JK uses ftp to transfer postscript-files from DOS to Unix.
- f9 graph: The postscript file is taken from DOS,
- f10 graph: and transferred to Unix.
- f11: JK using emacs for report-writing.
- f12 report: Retrieval of report from Unix file system.
- f13 report: Saving of report in Unix file system written in latex.
- f14: JK runs report through latex.
- f15 preliminary files: Retrieve latex file, postscript-figures and .bbl-file if any.
- f16 output files: Produced .dvi, .log, and .aux-files.
- f17: JK run report through bibtex.

- f18 bibliography: Retrieve .bib-file and .aux-file from Unix file system.
- f19 formatted bibliography: Produced .bbl-file.
- f20: JK runs dvips.
- f21 output file: .dvi-file.
- f22 postscript report: Report in .ps-format.

	From			*To*		
Name	Node	Role	Behalf	Node	Role	Behalf
f1	A102			A218		
f2	A218			A202		
f3	A102			A201		
f4	A201			A202		
f5	A102			A219		
f6	A219			A202		
f7	A202			A219		
f8	A102			A209		
f9	A202			A209		
f10	A209			A210		
f11	A102			A203		
f12	A203			A210		
f13	A210			A203		
f14	A102			A240		
f15	A240			A210		
f16	A210			A240		
f17	A102			A239		
f18	A239			A210		
f19	A210			A239		
f20	A102			A237		
f21	A237			A210		
f22	A210			A237		

The last figures show the interaction between different parts of the 'bibutil' actor introduced in Figure A.10. Figure A.14 illustrates the parts in members of the technical actor.

Actors

- A233 bibutil: Internal applicative.
- A245 constructhtml: Internal applicative. Creates a .html file to be used on WWW based on an already created bibliography-file.
- A246 constructitem: Internal applicative. Create a bibliography-file as a list of items based on an already created bibliography-file.
- A247 rhapend: Internal applicative. Perform wrap-up processing.
- A248 constructbib: Internal applicative. Create a latex-file with a bibliography.
- A249 createbib: Internal applicative. Process a latex file.
- A250 createcite: Internal applicative. Create a list of citations from a set of bibliography files, a category, and a keyword.
- A251 bsearch: Internal applicative. Create a bibliography based on the match on keywords.
- A252 rhapsody: Internal applicative. Create an up to date bibliography for the IS-group.
- A253 bibdef: Internal applicative. Front-end to define bibliography-entries.

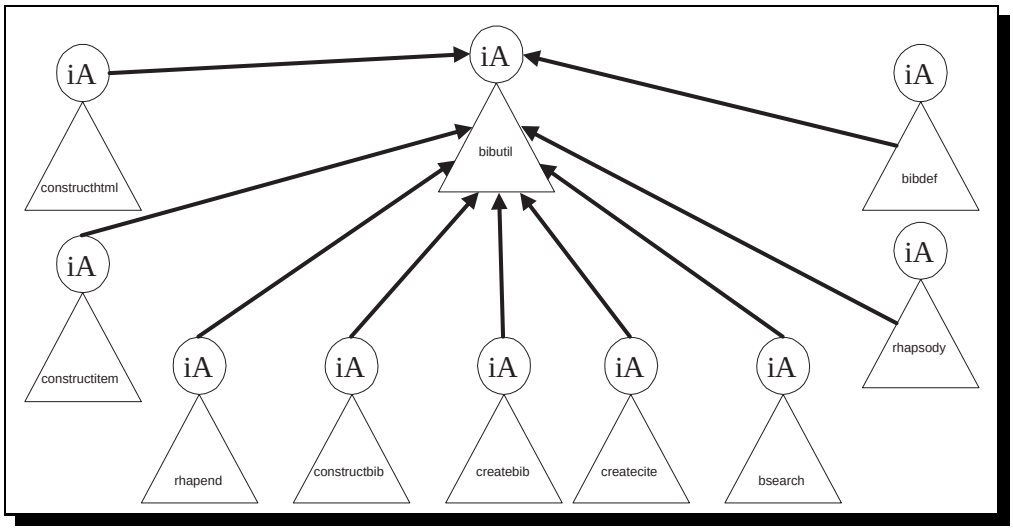

Figure A.14: Actor model describing the parts of the bibutil-actor

Relationship	From	Role	Behalf	To	Comment
Part-of	A245			A233	
Part-of	A246			A233	
Part-of	A247			A233	
Part-of	A248			A233	
Part-of	A249			A233	
Part-of	A250			A233	
Part-of	A251			A233	
Part-of	A252			A233	
Part-of	A253			A233	

In Figure A.15 we have illustrated how the control flows between different actors when a user uses the bibdef and bsearch applicative actors. We have here also used the PPM store-symbol. If only actors are used, these would be replaced by the SunOs-actor as in the previous example

Using bibdef, one enters bibliographic information, which is added to bibtex-files. The list of relevant files is updated if one indicate a new file. When searching for bibliography data, the user can call 'bsearch'. This will call the programs createcite, constructbib, and createbib. The user can also call createbib directly. Also the internal supportive actors latex, bibtex, and dvips are applied.

Actors

- A237 dvips: Internal supportive.
- A239 bibtex: Internal supportive.
- A240 latex: Internal supportive.
- A248 constructbib: Internal applicative.
- A249 createbib: Internal applicative.
- A250 createcite: Internal applicative.
- A251 bsearch: Internal applicative.
- A253 bibdef: Internal applicative.

Roles

- R135 User: Individual.

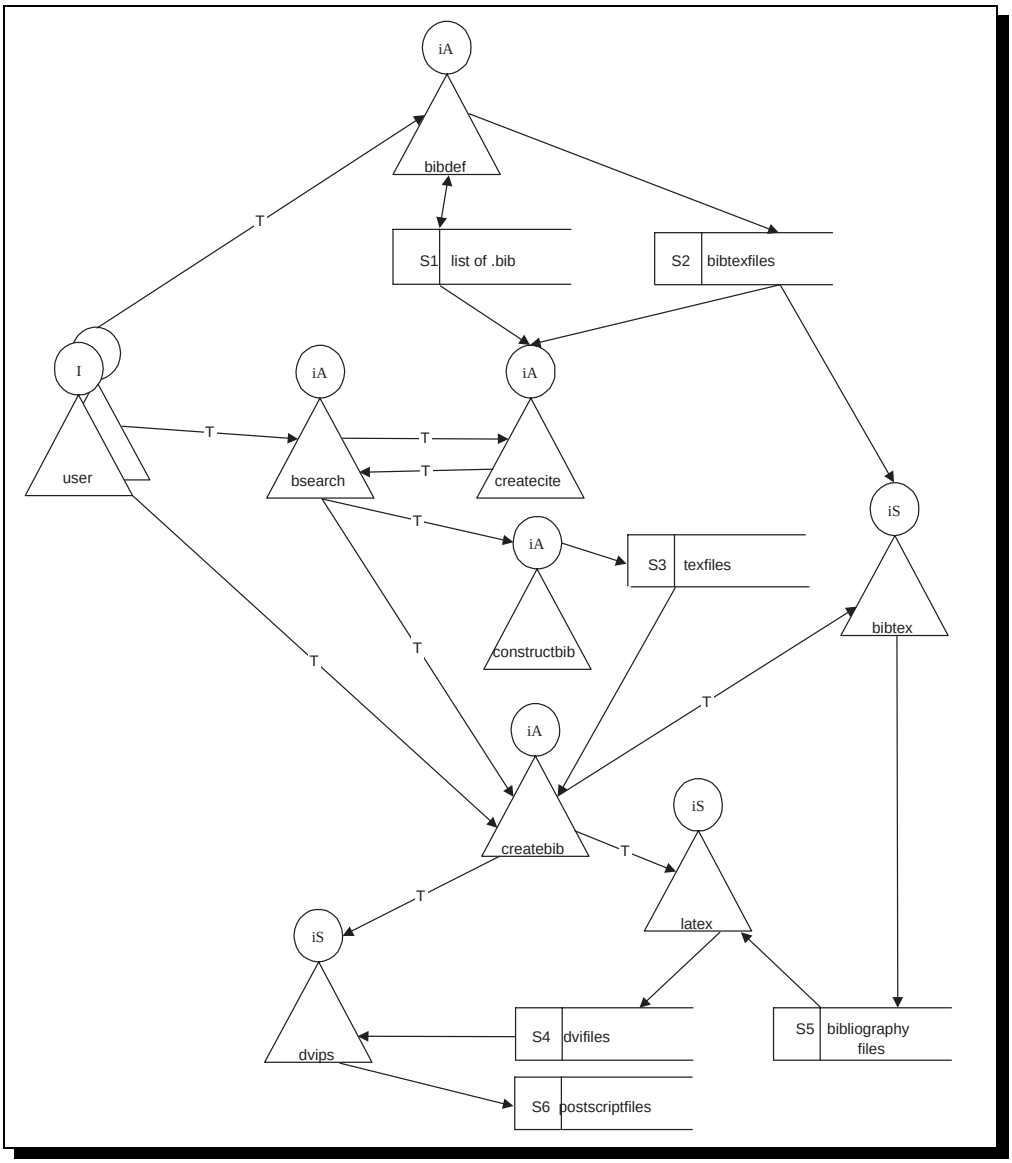

Figure A.15: Indicating control-flow in parts of bibutil

Flows

- f1: Run bibdef to insert new bibliography-entries..
- f2: Overview of existing bibliography-files.
- f3: Updates of the overview of bibliography-files.
- f4: Update of bibliography-file.
- f5: Start bsearch indicating keywords.
- f6: Run createcite to find bibliography-entries that match the keywords.
- f7: Take in the list of bibliography-file.
- f8: Take in the bibliographies.
- f9: Result from keyword-search.
- f10: Start construction of bibliography.
- f11: latex-file with bibliography to save.
- f12: Start the creation of the postscript-list.

- f13: Latex-file for processing.
- f14: Start bibtex on the latex file.
- f15: Bibliography-entries.
- f16: Resulting .bbl file.
- f17: Start createbib directly indicating already produced latex-file.
- f18: Start dvips to create postscript file.
- f19: Start latex to create .dvi-file.
- f20: Existing .bbl-file.
- f21: Created .dvi-file.
- f22: Existing .dvi-file
- f23: Resulting postscript-version of the bibliography.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R135		A235		
f2	A253			S1		
f3	S1			A235		
f4	A235			S2		
f5		R135		A251		
f6	A251			A250		
f7	S1			A250		
f8	S2			A250		
f9	A250			A251		
f10	A251			A248		
f11	A248			S3		
f12	A251			A249		
f13	S3			A249		
f14	A249			A239		
f15	S2			A239		
f16	A239			S5		
f17		R135		A249		
f18	A249			A237		
f19	A249			A240		
f20	S5			A240		
f21	A240			S4		
f22	S4			A237		
f23	A237			S6		

In Figure A.16 the call of the bsearch-part of the previous figure is depicted in PPM, with the extension that it is indicated which individual actors that are responsible for the processes. Here we are able to illustrate the control-flow of the system in more detail. First when a user starts bsearch, the optional parameters and mandatory keyword are checked. If OK, the keywords are send further to create a list of citations that matches the keywords, looking in the bibtex-files listed in the file-list. If not, an error-message is given. It is also possible to retrieve a help-text which explain how to use the program. When the citation-list has been created, a latex-file is created, and according to the parameters given, the citation-list is either given to the user, or a dvi-file is created. In the first case, the user can start the creation of the dvi-file himself. The dvi-file is made based on the tex-file and the bibtex files, and secondarily the postscript-files are created based on

the dvi-file by dvips. As we see, 'Sunos' is responsible for the store of postscript-files, and also all the other stores, although this is not indicated in the diagram.

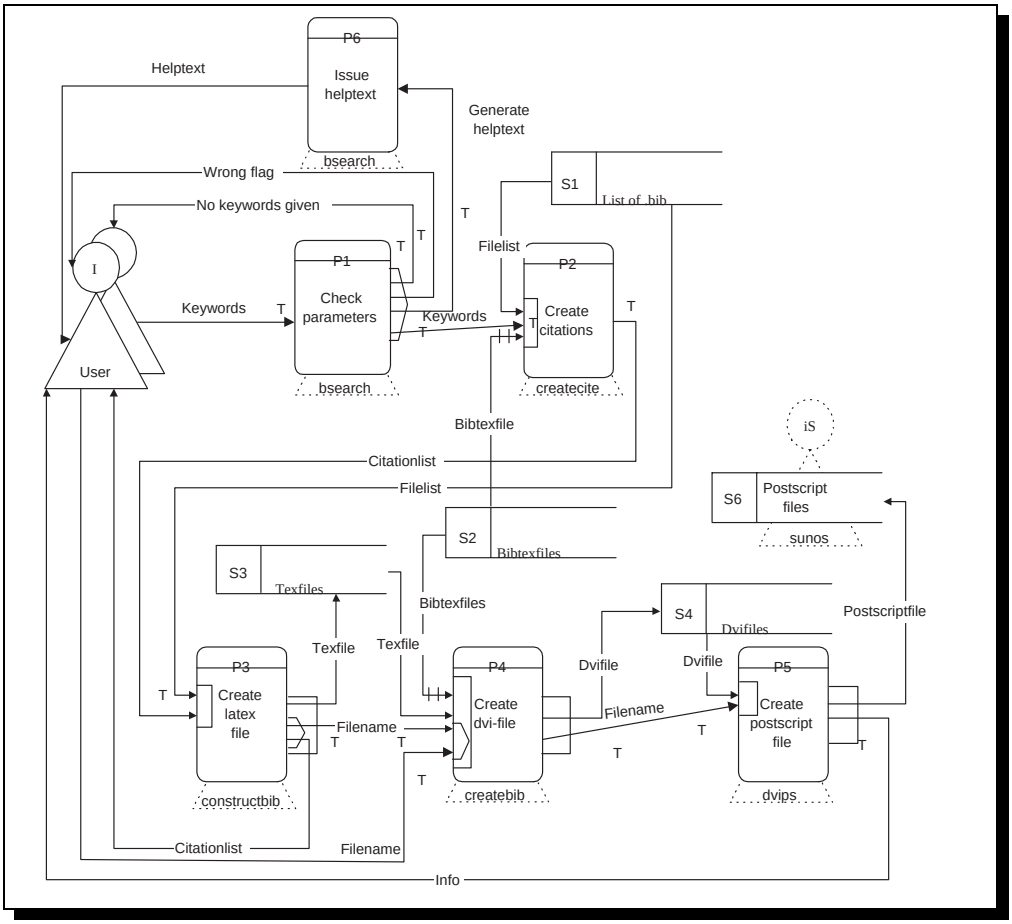

Figure A.16: PPM describing parts of bibutil

Actors

- A210 Sunos: Internal supportive.
- A237 dvips: Internal supportive.
- A248 constructbib: Internal applicative.
- A249 createbib: Internal applicative.
- A250 createcite: Internal applicative.
- A251 bsearch: Internal applicative.

Roles

- R135 User: Individual.

Stores

- S1 list of .bib.
- S2 bibtex files.
- S3 texfiles.
- S4 dvifiles.
- S6 Postscriptfiles.

Processes

- P1 Check parameters: Check if the given parameters are sufficient and legal. If not, an error-message is issued. If requested, a help-text is returned.
- P2 Create citations: Create the citations matching the given keywords.
- P3 Create latex file: Create a latex file including the retrieved citations.
- P4 Create dvi-file: Compile the latex-file.
- P5 Create postscript file.
- P6 Issue helptext.

Flows

- f1 keywords: Keywords and alternatively other parameters.
- f2 No keywords given: Error message.
- f3 Wrong flag: Wrong parameter given.
- f4 Generate help text.
- f5 Keywords.
- f6 Filelist: List of bibliography-files to search for the keywords.
- f7 Bibtexfile: Bibliography files.
- f8 Citationlist: List of citations indicating matching items.
- f9 Filelist: Same as f6.
- f10 Texfile: Latex-file including citations.
- f11 Filename: Name of created latex file.
- f12 Citationlist: Same as f8.
- f13 Bibtexfiles: Same as f7.
- f14 Texfile: Same as f10.
- f15 Filename: Same as f11.
- f16 Dvifile: Created dvi-file.
- f17 Filename: Name of dvi-file.
- f18 Dvifile: Same as f16.
- f19 Postscriptfile: Finished postscript-report with the extracted citations.
- f20 Info: Information about the created file.
- f21 Helptext: Canned help text for the system.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R135		P1		
f2	P1				R135	
f3	P1				R135	
f4	P1			P6		
f5	P1			P2		
f6	S1			P2		
f7	S2			P2		
f8	P2			P3		
f9	S1			P3		
f10	P3			S3		
f11	P3			P4		
f12	P3				R135	
f13	S2			P4		
f14	S3			P4		

continued on next page

continued from previous page						
	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f15		R135		P4		
f16	P4			S4		
f17	P4			P5		
f18	S4			P5		
f19	P5			S6		
f20	P5				R135	
f21	P6				R135	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A210			S1				
A210			S2				
A210			S3				
A210			S4				
A210			S6				
A237			P5				
A248			P3				
A249			P4				
A250			P2				
A251			P1				
A251			P6				

The link between actor-modeling and the other conceptual modeling languages will be further investigated in the conference case.

A.1.2 The Conference System

Our main example is a variant of the widely used "IFIP working conference" example. It has been used throughout the thesis, but not presented as a complete example. The example was proposed the first time in 1982 [243] and has been used to illustrate different modeling approaches, including different versions of the PPP-languages.

In this case, the example describes a real situation since we have been developing a system of this sort using many of the described languages. Thus the case-description is not the traditional description, but taken from an actual project. The conference in question was the IFIP WG8.1 Working Conference on Information Systems development for Decentralized Organizations (ISDO95). How the system was used in the preparations to this conference is described in [291].

The description of the case will be somewhat different from what is normal, since we instead of the presentation of one model, also present "intermediate" models, based on the view of only some of the participants.

After a general introduction of the case, project establishment and project preparation are described. Then models based on the three main users for the passive future image (i.e. the traditional situation as it is perceived for the organization of this kind of conference) are given, using actor-models and PPM. Based on these the generally agreed PPM model for the situation is given, and also the main speech acts being performed in conversations between an organizing committee of a conference and potential contributers of papers to such a conference. Process modeling is performed in the traditional way, and not centered around conversations.

Based on this and a limitation of the area of interest, a model of the perceived improved situation using PPM and ONER is presented. Based on this it is indicated which parts was supported by the application system in the first and the second release of the application system, and some of these processes are further described using DRL-rules and PLDs.

Rules and rule-hierarchies for both the current and future improved situation are given separately in the end of the case-description.

The models are based on [81], but also on intermediate models created as part of the project, meeting minutes from the project etc. They also look different because they originally used PPM as described in [116] in the project. The creation of rule-hierarchies was also done after the project was finished.

Overall problem description

IFIP is the acronym for International Federation for Information Processing. An IFIP working conference is an international conference that provides an opportunity for the computer scientists from IFIP member countries to discuss and interchange research results and new ideas on selected research fields.

The management of such a conference is usually done by two cooperating committees. The *program committee* (PC) handles the contents of the conference, say, the reviewing of papers, comprising sessions and tutorials, etc. The *organizing committee* (OC) handles the administration work, e.g. sending out invitations, registration of attendants, arranging time and places for sessions, dealing with financial matters, etc.

Project establishment

The project was established in the autumn of 1994, and had the following initial problem formulation, created by JK and AS in cooperation:

> "One are to create an application system to be used under the preparation, arrangement, and wrap-up in connection with scientific conferences and publication of scientific journals. It should be possible to use the system as is for the preparation and arrangement of the ISDO95 conference to be held in Trondheim in the autumn of 1995.
>
> The system should
>
> - Support electronic submission and treatment of papers.
> - Support the quality control of papers.
> - Support electronic distribution of information, and make information available on WWW.
> - Support the administration of the conference.
> - Support electronic access of the conference-proceedings, and other scientific articles for payment.
>
> The system should primarily be able to run on Unix workstations using X-windows, but should be extendible to run on PCs under Microsoft Windows."

Although some of the supporting actors of the system is indicating the initial problem-formulation, this did not mean that they were absolute requirements.

The first step of development started in September 1994, and since the system was developed by a group of students as part of their education, the absolute deadline for the first version of the application system was December 1. 1994. In addition, the total hourly budget was limited to approximately 1500 hours. Based on this, and the potential wide scope of the overall task, it was early decided to use incremental development, by performing a broad analysis of the situation, but only implement a small part of this in the first release. If the result looked promising, a second version should be prepared and be ready in the beginning of February 1995.

The project group consisted of 7 persons, JPH, YJ, KPH, HFS, SG, EA, and FVL. They were all fourth year students at NTH, with comparable academic background. Although their background on other areas differed substantially, they will in the rest of the description be refereed to as the project group, or *GR* for short.

An overview of the organization of the group is given in Figure A.17, including the main roles that was institutionalized by the group. The project group is the top internal actor in the diagram. The project group is a temporary organizational actor. The figure also show the institutionalized communication with additional actors, and in which roles they are contacted. Note that these roles are internal roles to the periodic organizational actor of the course in which this project took place. Also the main supporting technical actors supporting the project-group are shown in the top of the model.

Actors

- A34 Project group 4: Internal organizational.
- A101 AS : External Individual.
- A102 JK : External Individual.
- A133 JPH: Internal individual.
- A134 YJ: Internal individual.
- A135 KPH: Internal individual.
- A136 HFS : Internal individual.
- A137 SG : Internal individual.
- A138 EA : Internal individual.

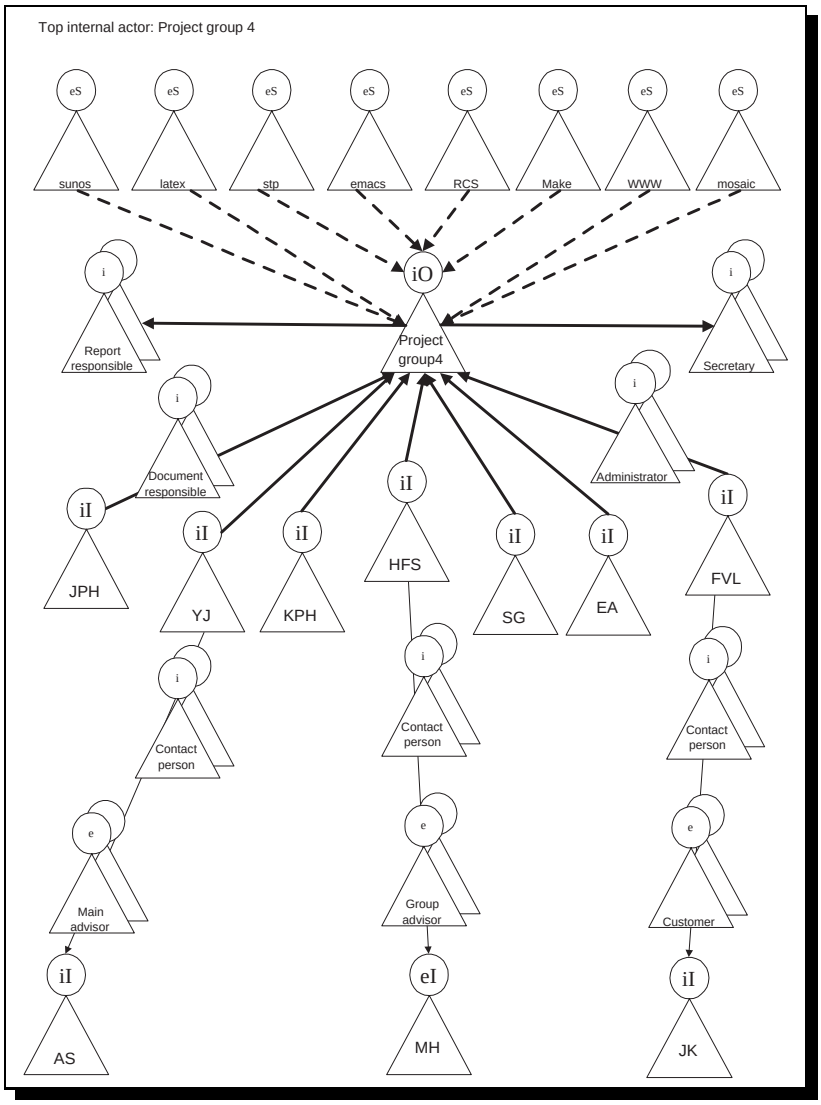

Figure A.17: Actor model describing actors and roles in the project-group

- A139 FVL : Internal individual.
- A140 MH : External individual.
- A203 emacs: External software. Text editor.
- A210 Sunos: External software. Operating system.
- A212 Mosaic: External software. Web client.
- A225 WWW: External software. Web server.
- A227 make: External software. Building tool.
- A228 RCS: External software. Version control.
- A240 latex: External software.
- A254 stp: External software. CASE tool.

Roles

- R136 Report responsible: Internal individual. Responsible for creating templates for reports, meeting-minutes, status-reports and meeting-calls. He is also responsible for assuring that standards are adhered to. He is responsible for printout, distribution, and backup

of documents, and to keep overview of who has borrowed books and other material.

- R137 Document responsible: Internal individual. Each group member have overall responsibility for one of the reports that are to be produced during the project.
- R138 Administrator: Internal individual. Responsible for having the overall control of the project. He is responsible for that the project plan is followed, and for the preparation of meeting- agendas and meeting-calls, and the weekly status report. He is also responsible for assuring that the work is divided evenly on the group members.
- R139 Secretary: Internal individual. Everyone except for the administrator function as secretary during the project. The secretary is responsible for creating a minute from all meetings, and that the minutes are available at the latest 24 hours after the meeting took place.
- R140 Contact person: Internal individual.
- R141 Main advisor: External individual.
- R142 Group advisor: External individual.
- R143 Customer: External.

Relationship	From	Role	Behalf	To	Comment
Part-of	A133	R137		A34	
Part-of	A134			A34	
Part-of	A135			A34	
Part-of	A136			A34	
Part-of	A137			A34	
Part-of	A138			A34	
Part-of	A139	R138		A34	
Instituted-by	R136			A34	
Instituted-by	R139			A34	

Flows

- f1: Messages from the group to AS as main advisor in the project.
- f2: Messages from the group to MH as group advisor in the project.
- f3: Messages from the group to JK as customer in the project.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	A134	R140		A101	R141	
f2	A136	R140		A140	R142	
f3	A139	R140		A102	R143	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A203			A34				
A210			A34				
A212			A34				
A225			A34				
A227			A34				
A228			A34				
A240			A34				
A254			A34				

Preparation for modeling

To identify stakeholders, a preliminary actor-model of the situation was developed as illustrated in Figure A.18. The internal organizational actor in this case is IDT, thus the conference organization as such is an external temporary organizational actor. This is part of TC8, which again is part of IFIP, which, as indicated both have many other members.

The conference organization consist in this case of a program committee, consisting of a set of PC-members. The PC-members are not included in the diagram. The PC members came from four continents, mainly from Europe.

In addition the program committee chair is part of the program committee. The organizing committee, consisted of four persons internal to IDT, the program chair, the organizing chair, the program coordinator, and a managing clerk. Finally, the general chair was member of the overall organization.

Certain rules apply to the different roles of such an organization although they might be different for different organizations arranging conferences, such as ACM, IEEE, and IFIP, not to mention individual conference series such as VLDB and CAiSE, and the actual roles and number of roles and persons in the different roles might differ according to the conference in question. The roles are usually institutionalized by a more permanent organization, in this case IFIP TC8 which have issued policies and guidelines for the planning and organizing of Working Conferences. In Figure A.18 we have also indicated the publisher of the proceedings, Chapman & Hall as supporting the organizing committee in the publishing of the proceedings through an agreement with IFIP. SEVU is the local organization for arranging conferences at NTH, and support the organizing chair.

Actors

- A1 NTH: External organizational. The Norwegian Institute of Technology.
- A35 IFIP: External organizational.
- A36 IFIP TC8: External organizational. Technical committee within IFIP.
- A22 ISDO95: External organizational.
- A37 ISDO PC: External organizational. Program committee of ISDO95.
- A38 ISDO OC: External organizational. Organizing committee of ISDO95.
- A39 SEVU: External organizational.
- A40 Chapman & Hall: External organizational.
- A101 AS: Internal individual.
- A102 JK: Internal individual.
- A115 OIL: Internal individual.
- A125 BA: Internal individual.
- A141 KJ: External individual.

Roles

- R3 Publisher: External organizational. Publisher of proceedings of a conference.
- R4 Conference organizer: External organizational. Specialist on the practical arrangements of conferences.
- R108 PC-member: External individual. For PC-members, the following rules often applies:
  - It is obligatory for a PC-member to distribute the call for paper widely before the CFP-deadline.
  - It is obligatory for a PC-member to distribute the call for participation widely before the conference.

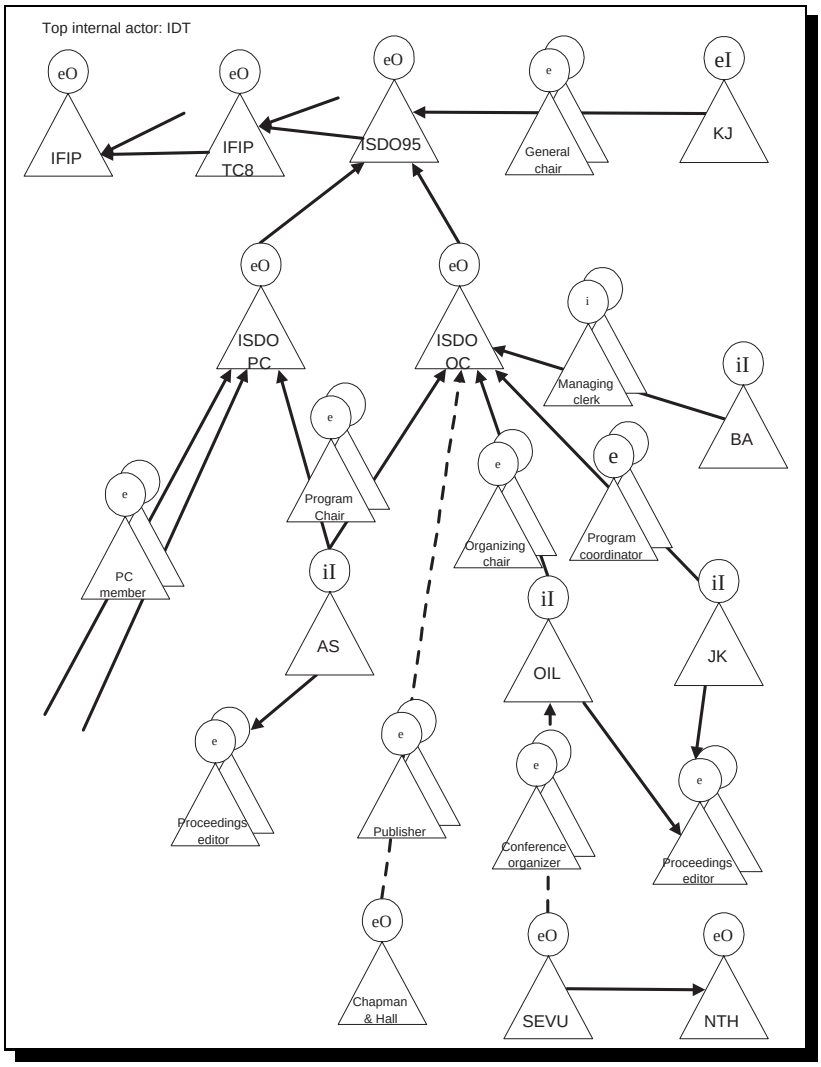

Figure A.18: Actor model describing the conference organization

  - It is obligatory for a PC-member to review the papers that have been distributed to him/her and send the reviews to the organizing committee before the review deadline.
  - It is recommended for a PC-member to attend the program committee meeting.
  - It is recommended for a PC-member to attend the conference.

- R112 Program coordinator: External individual. The responsibilities of this role is often part of the program committee chair-role. Includes the soliciting and processing of papers.
- R128 Managing clerk: Internal individual.
- R144 General chair: External individual. Has overall responsibility for all conference matters. Is responsible for overall budget planning and control. Set plan and control milestones.
- R145 Proceedings editor: Internal individual. Responsible for editing and publishing of the conference proceedings.
- R146 Program chair: External individual. Responsible for the technical program in general, including the recruiting of PC-members. Recommendations to the PC-chair on who to invite are:

  - It is recommended for the PC-chair to invite persons that are not over-committed, and

who are capable and willing to perform a high-quality reviewing job on schedule.
  - It is recommended for the PC-chair to achieve a good geographical distribution among the PC-members.
  - It is recommended for the PC-chair to involve some persons that have not been in these types of program-committees before.
  - It is recommended for the PC-chair to involve people that are knowledgeable and have contacts in the field of the conference.

  This role is also responsible for preparing the call for paper and establishing reviewing and selection procedures for the technical content of the program. During the conference, he is responsible for briefing session chairmen on their duties and for ensuring that the session chairmen and the speakers have a chance to meet prior to the session.
- R147 Organizing chair: External individual. Responsible for all local arrangements. Responsible for publicity and the social program. Responsible for all matters relating to the smooth running of the conference. Responsible for registration and all financial matters.

Relationship	From	Role	Behalf	To	Comment
Part-of	A22			A36	
Part-of	A36			A35	
Part-of	A37			A22	
Part-of	A38			A22	
Part-of	A39			A1	
Part-of	A101	R146		A37	
Part-of	A101	R146		A38	
Part-of	A102	R112		A38	
Part-of	A115	R147		A38	
Part-of	A125	R128		A38	
Part-of	A141	R144		A22	
Part-of	ANNN			A35	Many
Part-of	ANNN			A36	Many
Part-of	ANNN	R108		A37	Many
Fill	A101			R145	
Fill	A102			R145	
Fill	A115			R145	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A39	R4		A115				
A40	R3		A38				

Additional stakeholders in this situation, apart from the project group itself, were those receiving call for papers and call for participation, and the subset of these being the contributors and/or conference attendants. For an extended project which would include making the system into a share-ware solution, several other potential stakeholders, i.e. potentially future program and organizing committees and different conference arrangement organization and publishers could be identified.

In addition, JK and OIL would be the users of the system, JK being the prime direct user, at least of the first version of the system. AS also functioned as a provider of information. Three distinct realities could be distinguished. The system developer group *GR*, a mature professor *AS* with long experience including the participation in many program committees and conference organizing efforts and two young researchers connected to the IS-group, *JK* and *OIL*. Although having different requirements to a system of this sort, both had a comparable background, and had similar experience with this kind of conference, i.e. mainly as contributors and participants, in addition to having experience from the practical arrangement of similar seminars.

CATWOE analysis

As a starting point for modeling, the root-definitions [51] for all the participants were established using the CATWOE-technique (see Chapter 4).

The question

> **Who** is doing **what** for **whom**, and to whom are they **answerable**, what **assumptions** are being made, and in what **environment** is this happening?

was divided into its parts, and was answered by the social members of the audience as indicated below:

OIL, organizing chair:

Customer (the whom): The organizing chair of the conference and SEVU.

Actor (the who): GR and JK.

Transformation (the what): Good communication and coordination between SEVU and the organizing chair.

Weltanschauung (the assumptions): A well-organized conference is of major importance to the research group.

Owner (the answerable): AS.

Environment: The university.

JK, program coordinator:

Customer (the whom): JK.

Actor (the who): GR.

Transformation (the what): Possibility to support electronic submission and review of papers.

Weltanschauung (the assumptions): As program coordinator I have to make sure that people know about the conference, issues papers, and that the papers are given a proper review.

Owner (the answerable): AS.

Environment: IS-group.

AS program chair:

Customer (the whom): IS-group.

Actor (the who): GR, JK.

Transformation (the what): The creation of a general purpose conference system which can also be applied by others.

Weltanschauung (the assumptions): Active research groups should arrange conferences.

Owner (the answerable): AS.

Environment: IFIP8.1.

GR, project group:

Customer (the whom): JK.

Actor (the who): GR.

Transformation (the what): Improve the organization of the ISDO95 conference by creating a general system for conference support.

Weltanschauung (the assumptions): We want the customer to be satisfied with the product, so we will get a best possible grade on the project.

Owner (the answerable): AS.

Environment: Project course at IDT.

As we see when comparing the above with how the roles are usually defined, we find certain local discrepancies. The further modeling work is based upon how the participants perceived their role, and not general guidelines from external organizations. We also notice certain discrepancies among the world-views especially on who is the customer, the transformation that are to take place, and the environment.

We can identify one more actor in addition to the preliminary actor-diagram namely the project course, being a periodic organizational actor. A model indicating this actor as the top internal organizational one is given in Figure A.19. This project course is described in more detail in [6]. The model indicates the situation in the autumn of 1994. Although the involved actors often change from year to year, the internal roles are similar. As we see, JK is one of several customers (or rather customer representatives).

Actors

- A3 IDT: External organizational.
- A34 Project group 4: Internal organizational.
- A41 Group project: Internal organizational.
- A42 Project group 1: Internal organizational.
- A43 Project group 2: Internal organizational.
- A44 Project group 3: Internal organizational.
- A45 Project group 5: Internal organizational.
- A46 Project group 6: Internal organizational.

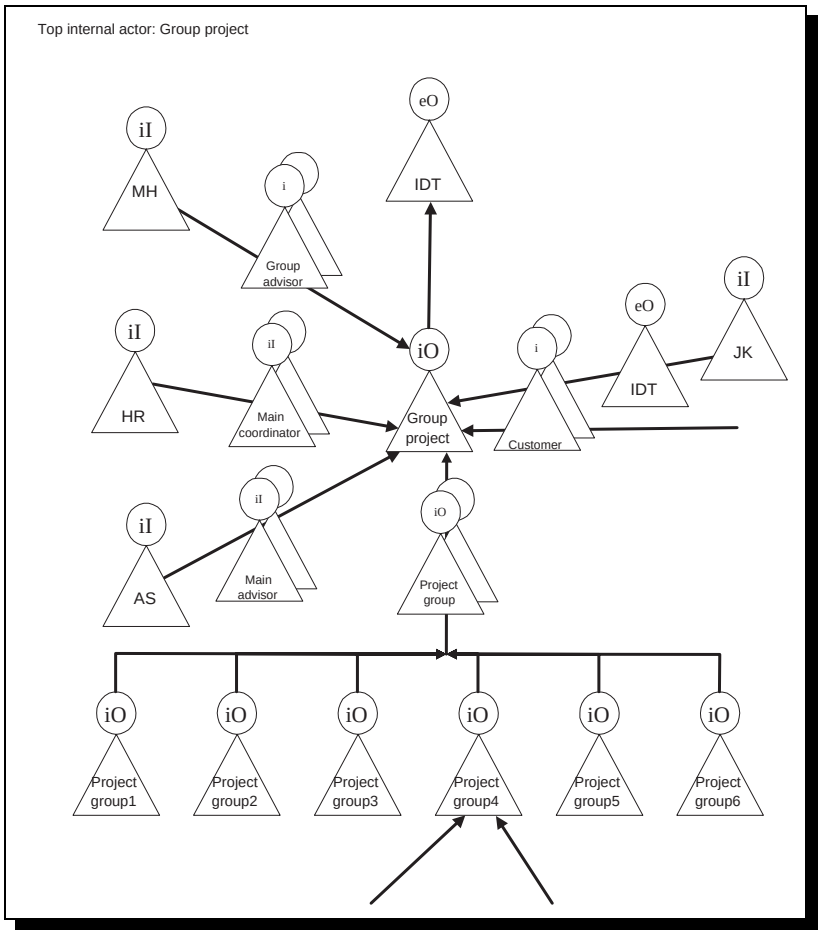

Figure A.19: Actor model describing the group-project

- A101 AS : Internal individual.
- A102 JK : Internal individual.
- A118 HR: Internal individual.
- A140 MH : Internal individual.

Roles

- R4 Project group: Internal organizational.
- R141 Main advisor: Internal individual.
- R142 Group advisor: Internal individual.
- R143 Customer: Internal.
- R148 Main coordinator: Internal individual.

Relationship	From	Role	Behalf	To	Comment
Part-of	A34	R4		A41	
Part-of	A41			A3	
Part-of	A42	R4		A41	
Part-of	A43	R4		A41	
Part-of	A44	R4		A41	
Part-of	A45	R4		A41	
Part-of	A46	R4		A41	
Part-of	A101	R141		A41	
Part-of	A102	R143	A3	A41	
Part-of	A118	R148		A41	
Part-of	A140	R142		A41	
Part-of	ANNN	R143		A41	According to the number of groups

Model of current situation/passive future image

This modeling was based on three different internal realities, those of JK, AS, and OIL, and also regarded different, but overlapping areas for each actor as illustrated below. In the modeling of the current situation, it is used informal rules, rule-hierarchies and process modeling included actors and actor-support. The overall model that the project-group developed based on this, is similar to the models traditionally developed for this kind of events, and is highly influenced by earlier developed models of the situation.

JK functioned as the primary source of knowledge about the overall paper process, not including details about the PC-meeting. AS functioned as the primary source for the overall process, with special focus on the work of a program committee. OIL functioned as the primary source of information on the practical organization of the conference and contact with a conference organizers.

Creation and distribution of Call for Papers :

The diagram is given in Figure A.20. The main source of the model was *JK* . It describes the creation and distribution of the call for papers. Internal and external is below relative to a conference OC.

Roles

- R112 Program coordinator : Internal individual.
- R146 Program chair : Internal individual.
- R149 Contributer: External individual. Someone potentially contributing a paper to the conference.

Stores

- S1 Call for papers: Contains the call for papers to be distributed.
- S2 Address list: List of addresses for those that are to receive the call for papers.

Timers

- C1 Clock: Trigging the creation of call for paper.
- C2 Clock: Trigging the issuing of the CFP.

Processes

- P1: Make CFP: The CFP is created.
- P2: Distribute CFP. The CFP is issued, either to all the members of the address list, or to a single person. Neither can be done before the CFP is created.

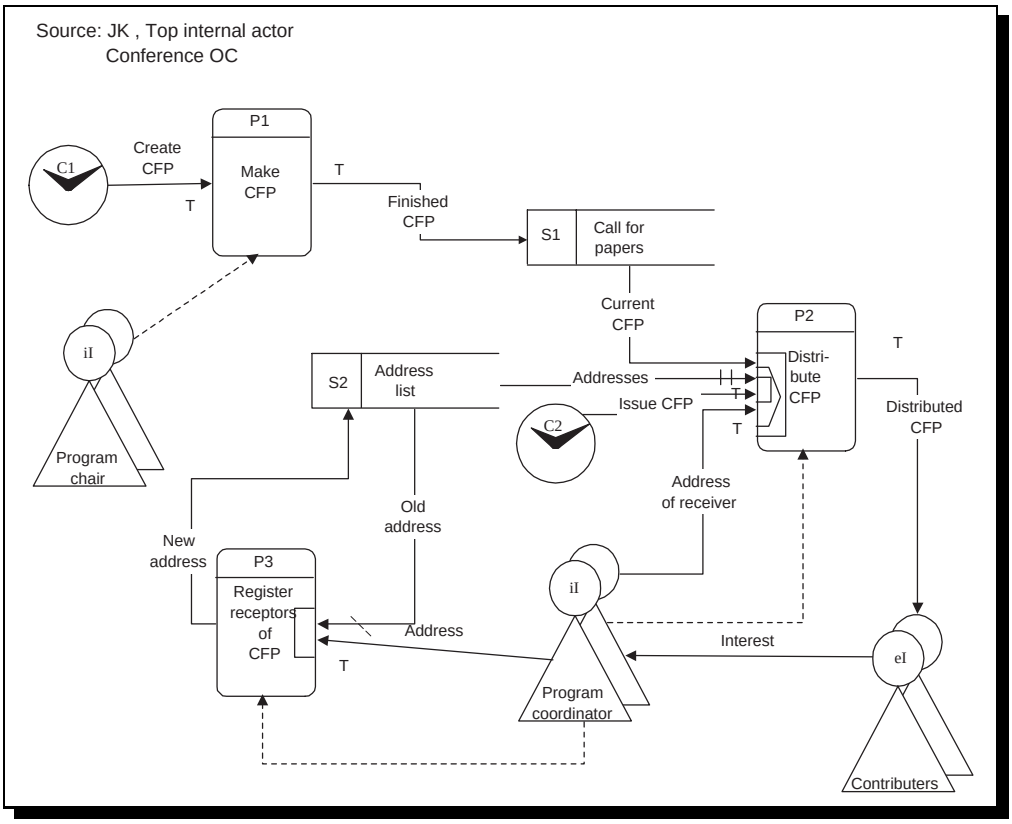

Figure A.20: PPM describing creation and distribution of CFP

- P3: Register receptors of CFP: A process to update the address list, based on the existing address-list and new address-data.

Flows

- f1 Create CFP: Signal to start the creation of the CFP.
- f2 Finished CFP: The finished call for paper.
- f3 Current CFP.
- f4 Addresses: The address list.
- f5 Issue CFP: Signal to start the issuing of the CFP.
- f6 Address of receiver: Address to send the CFP to.
- f7 CFP: CFP sent to contributers.
- f8 Interest: Notification of interest in the conference from a possible contributer.
- f9 Address: Address of potential contributer.
- f10 Old address: Address in the address-register that is updated.
- f11 New address: New or updated address.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	C1			P1		
f2	P1			S2		
f3	S1			P2		
f4	S2			P2		
f5	C2			P2		
f6		R112		P2		
f7	P2				R149	
f8		R149			R112	
f9		R112		P3		
f10	S2			P3		
f11	P3			S2		

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
	R112		P2				
	R112		P3				
	R146		P1				

Reception and distributions of papers :

The diagram is given in Figure A.21. Main source: JK. Top internal actor: Conference OC. When a paper is received, one control that sufficient data about the authors and the paper is also received. If not one asks for further information. One also send a confirmation to the contributer that the paper is received. After the paper is registered, it is distributed to PC-members for review.

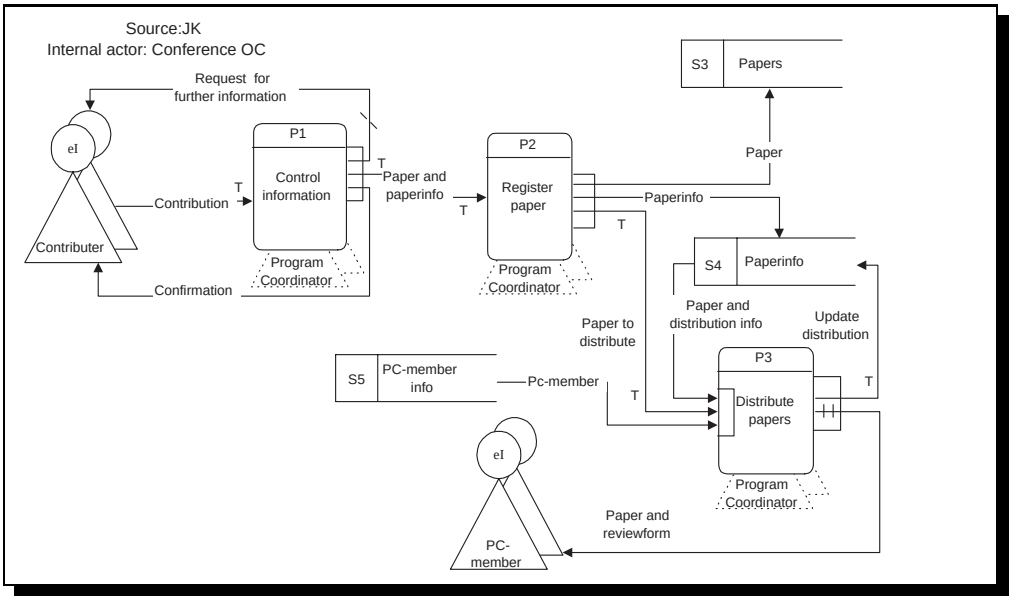

Figure A.21: PPM describing reception and distributions of papers

Roles

- R108 PC member: External individual.
- R112 Program coordinator : Internal individual.
- R149 Contributer: External individual. Someone potentially contributing a paper to the conference.

Stores

- S3 Papers: Received papers.
- S4 Paperinfo.
- S5 PC-member info.

Processes

- P1 Control information: Check that all necessary data about the paper and authors are issued together with the paper. If not ask the contributer for the missing data.
- P2 Register paper.
- P3 Distribute papers. Distribute the paper to review among the members of the program committee.

Flows

- f1 Contribution: Paper submitted to the conference.
- f2 Confirmation: Confirmation of reception of paper.
- f3 Request for further information: Sent together with the confirmation in case some data about the paper or authors of the paper is missing.
- f4 Paper and paperinfo: Sent to registration.
- f5 Paper: A physical paper.
- f6 Paperinfo: Information about a paper.
- f7 Paper to distribute. A copy of the paper to distribute to reviewers.
- f8 Paper and distribution info. Information to use when deciding who should be given the paper to review.
- f9 PC-member: Information about PC-members.
- f10 Paper and review form.
- f11 Update distribution: After distributing the paper, update the overview to indicate that the paper has been distributed for review.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R149		P1		
f2	P1				R149	
f3	P1				R149	
f4	P1			P2		
f5	P2			S3		
f6	P2			S4		
f7	P2			P3		
f8	S4			P3		
f9	S5			P9		
f10	P3				R108	
f11	P3			S4		

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
	R112		P1				
	R112		P2				
	R112		P3				

Reception of CRC :

The diagram is given in Figure A.22. Main source: JK. Top internal actor: Conference OC. When a CRC is received, it is checked that it is according to the style-guide and the comments by the reviewers. If not, the contributer is asked to issue a new version. A week before the CRC-deadline, a reminder is issued to those with accepted papers who have not yet submitted their CRC. When all CRCs are received, the manuscript for the proceedings is put together and sent to the publisher.

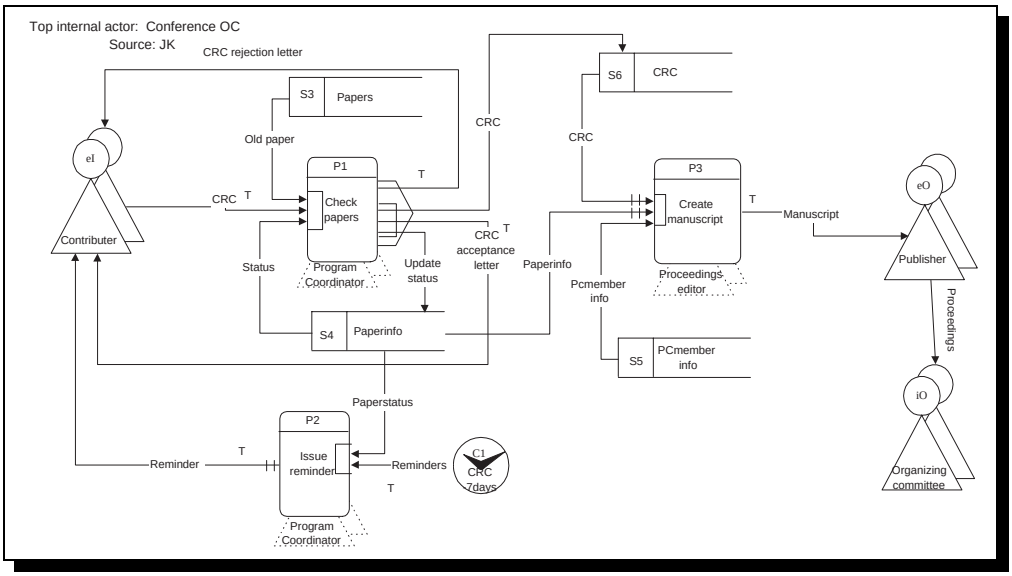

Figure A.22: PPM describing reception of CRC

Roles

- R5 Publisher: External organizational. Publisher of e.g. proceedings.
- R6 Organizing committee: Internal organizational.
- R112 Program coordinator : Internal individual.
- R145 Proceedings editor:
- R149 Contributer: External individual. Someone potentially contributing a paper to a conference.

Stores

- S3 Papers: Received papers.
- S4 Paperinfo.
- S5 PC-member info.
- S6 CRC: Camera-ready copy of papers.

Timers

- C1 CRC - 7 days: Send a signal to start the sending of reminders a week before the deadline for CRC to those not yet having issued this.

Processes

- P1 Check papers: Check that the CRC are according to the guidelines given for the format and content of the CRC.
- P2 Issue reminder: Issues a reminder to those not having sent in the CRC of their accepted paper.
- P3 Create manuscript. Put together the manuscript for the proceedings and send it to the publisher.

Flows

- f1 CRC: Camera ready copy of an accepted paper.
- f2 Old paper: The originally accepted paper.
- f3 Status: If the paper is conditionally accepted or not.
- f4 CRC rejection letter: Sent if the CRC is not according to the guidelines.
- f5 CRC: Storing the CRC before creating the manuscript.
- f6 CRC acceptance letter: A confirmation that the CRC is OK.
- f7 Update status: Update the status of the paper indicating that the CRC has been received.
- f8 CRC: CRC to be used as part of the proceedings.
- f9 Paperinfo: Information about paper used for the creation of table of contents.
- f10 PCmember info. Information about the organizers of the conference to be included in the preface of the proceedings.
- f11 Manuscript: Manuscript of the proceedings.
- f12 Reminders: Start issuing reminders to contacts of accepted papers.
- f13 Paperstatus: Status of the accepted papers.
- f14 Reminder: A reminder that it is time to issue the CRC.
- f15 Proceedings: The printed proceedings.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R149		P1		
f2	S3			P1		
f3	S4			P1		
f4	P1				R149	
f5	P1			S6		
f6	P1				R149	
f7	P1			S4		
f8	S6			P3		
f9	S4			P3		
f10	S5			P3		
f11	P3				R5	
f12	C1			P2		
f13	S4			P2		
f14	P2				R149	
f15		R5			R6	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
	R112		P1				
	R112		P2				
	R145		P3				

Distributions of papers to reviewers :

The diagram is given in Figure A.23. Main source: AS. Top internal actor: Conference OC. When all papers have been received, one decide the number of reviewers according to the number of papers and PC-members. If more papers than expected have been received, one might need to invite further members to the program-committee. When the number of reviewers are decided, it is decided which papers should be given to which reviewers, and they are distributed together with a review-form to the PC-members.

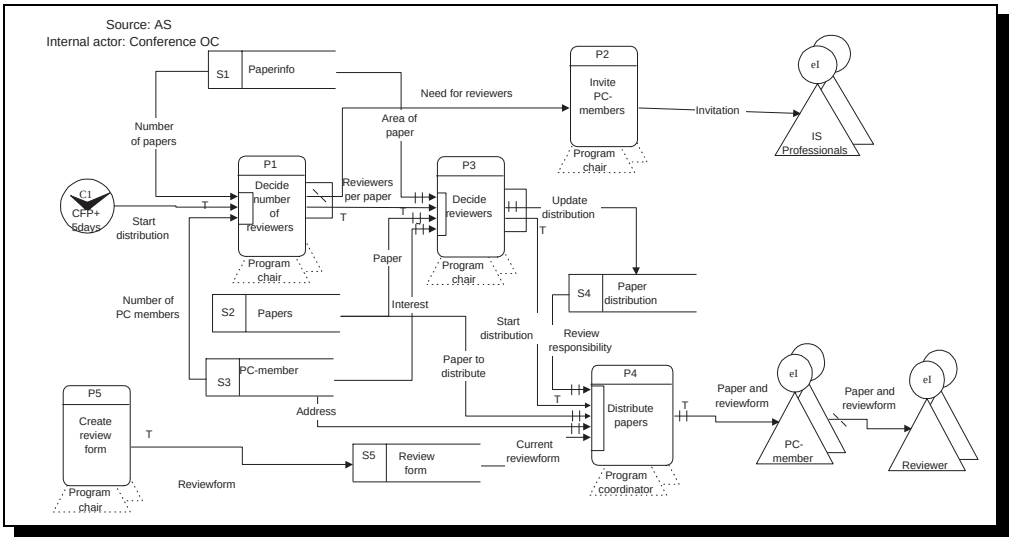

Figure A.23: PPM describing distribution of papers to reviewers

Roles

- R108 PC member: External individual.
- R112 Program coordinator : Internal individual.
- R146 Program chair : Internal individual.
- R150 IS professional: External individual. A potential member of the program committee.
- R151 Reviewer: External individual. A reviewer not part of the program committee.

Stores: Note that the numbering is different here than above, since the source of the models is another actor.

- S1 Paperinfo.
- S2 Papers.
- S3 PC-member.
- S4 Paper distribution..
- S5 Review form.

Timers

- C1 CFP + 5 days. Five days after the deadline for issuing papers.

Processes

- P1 Decide number of reviewers: Based on the number of papers and the number of PC-members, the number of reviewers of each paper is decided. It might happen that one find that one need extra PC-members if the number of submitted papers are much higher than expected.
- P2 Invite PC-members: Send an invitation to IS-professionals to join the program-committee of the conference.
- P3 Decide reviewers: Decide which reviewers should review which papers.
- P4 Distribute papers: According to the result in process P3.
- P5 Create review form: Make the review form that the reviewers should use in the review.

Flows

- f1 Decide reviewers: Signal to start the process of deciding reviewers.
- f2 Number of papers.
- f3 Number of PC-members.
- f4 Need for reviewers.
- f5 Reviewers per paper.
- f6 Area of paper: Keywords indicating the subject of the paper.
- f7 Paper: The submitted paper.
- f8 Interest: Keywords indicating the interest areas of the PC-members.
- f9 Update distribution.
- f10 Start distribution: Signal.
- f11 Reviewform.
- f12 Current reviewform.
- f13 Invitation: Invitation to join the program committee.
- f14 Review responsibility: Overview of the papers a given PC-member is set to review.
- f15 Paper to distribute: Copy of submitted paper.
- f16 Address: Address of PC-member.
- f17 Paper and reviewform.
- f18 Paper and reviewform.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	C1			P1		
f2	S1			P1		
f3	S3			P1		
f4	P1			P2		
f5	P1			P3		
f6	S1			P3		
f7	S2			P3		
f8	S3			P3		
f9	P3			S4		
f10	P3			P4		
f11	P5			S5		
f12	S5			P4		
f13	P2				R150	
f14	S4			P4		

continued on next page

continued from previous page						
	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f15	S2			P4		
f16	S3			P4		
f17	P4				R108	
f18		R108			R151	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
	R112		P4				
	R46		P1				
	R46		P2				
	R46		P3				
	R46		P5				

Selection of papers :

The diagram is given in Figure A.24. Main source: AS. Top internal actor: Conference OC. Reviews are registered as they are received. A reminder is issued at the deadline for the review to all who have not yet submitted all their reviews. Papers are selected based on these reviews in a program committee meeting, and the result is issued to the contributer right after the meeting is finished.

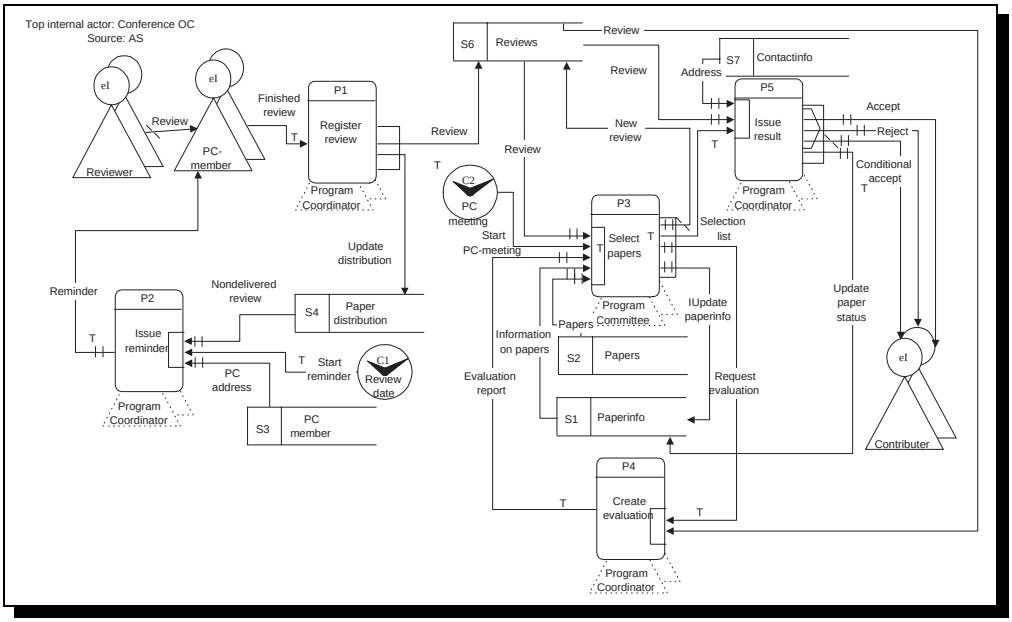

Figure A.24: PPM describing selection of papers

Roles

- R7 Program committee: External organizational.

- R108 PC member: External individual.
- R112 Program coordinator : Internal individual.
- R149 Contributer: External individual. Someone potentially contributing a paper to the conference.
- R151 Reviewer: External individual. A reviewer not part of the program committee.

Stores

- S1 Paperinfo.
- S2 Papers.
- S3 PC-member.
- S4 Paper distribution.
- S6 Reviews.
- S7 Contactinfo.

Timers

- C1 Review date: Deadline for sending in review.
- C2 PC-meeting: Time of PC-meeting.

Processes

- P1 Register review.
- P2 Issue reminder: Issue reminder on reviews when the deadline for reviews is passed.
- P3 Select paper: Select the papers to be presented at the conference.
- P4 Create evaluation: Create evaluation reports based on the reviews of papers.
- P5 Issue result: Issue the result of the selection process at the program committee meeting to the contributers.

Flows

- f1 Review: Preliminary review.
- f2 Finished review.
- f3 Review: Registered review.
- f4 Update distribution: Indicate that the review has been received.
- f5 Start PC-meeting: Signal.
- f6 Review.
- f7 Evaluation report: Based on reviews.
- f8 Information on papers.
- f9 Papers.
- f10 New reviews: In case one have received few or very varying reviews on a paper, a review is made specifically during the PC-meeting.
- f11 Selection list: The list of accepted papers.
- f12 Request evaluation: A request for an evaluation report.
- f13 Update paperinfo.
- f14 Review.
- f15 Address: Address of paper-contact.
- f16 Accept: Notification of acceptance of paper.
- f17 Reject: Notification of rejection of paper.
- f18 Conditional accept: Notification of conditional accept of paper.
- f19 Update paperstatus: Update status of papers according to if it is accepted or not.
- f20 Review: Review used as basis for the creation of evaluation report.
- f21 Nondelivered review: A list of the missing reviews.
- f22 Start reminder: Signal.
- f23 PC address.

- f24 Reminder.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R151			R108	
f2		R108		P1		
f3	P1			S6		
f4	P1			S4		
f5	C2			P3		
f6	S6			P3		
f7	P4			P3		
f8	S1			P3		
f9	S2			P3		
f10	P3			S6		
f11	P3			P5		
f12	P3			P4		
f13	P3			S1		
f14	S6			P5		
f15	S7			P5		
f16	P5				R149	
f17	P5				R149	
f18	P5				R149	
f19	P5			S1		
f20	S6			P4		
f21	S4			P2		
f22	C1			P2		
f23	S3			P2		
f24	P2				R108	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
	R7		P3				
	R112		P1				
	R112		P2				
	R112		P4				
	R112		P5				

Interactions in connection with conference organization :

The diagram is given in Figure A.25. Main source: OIL, but also based on conversation with SEVU. Top internal actor is a conference OC.

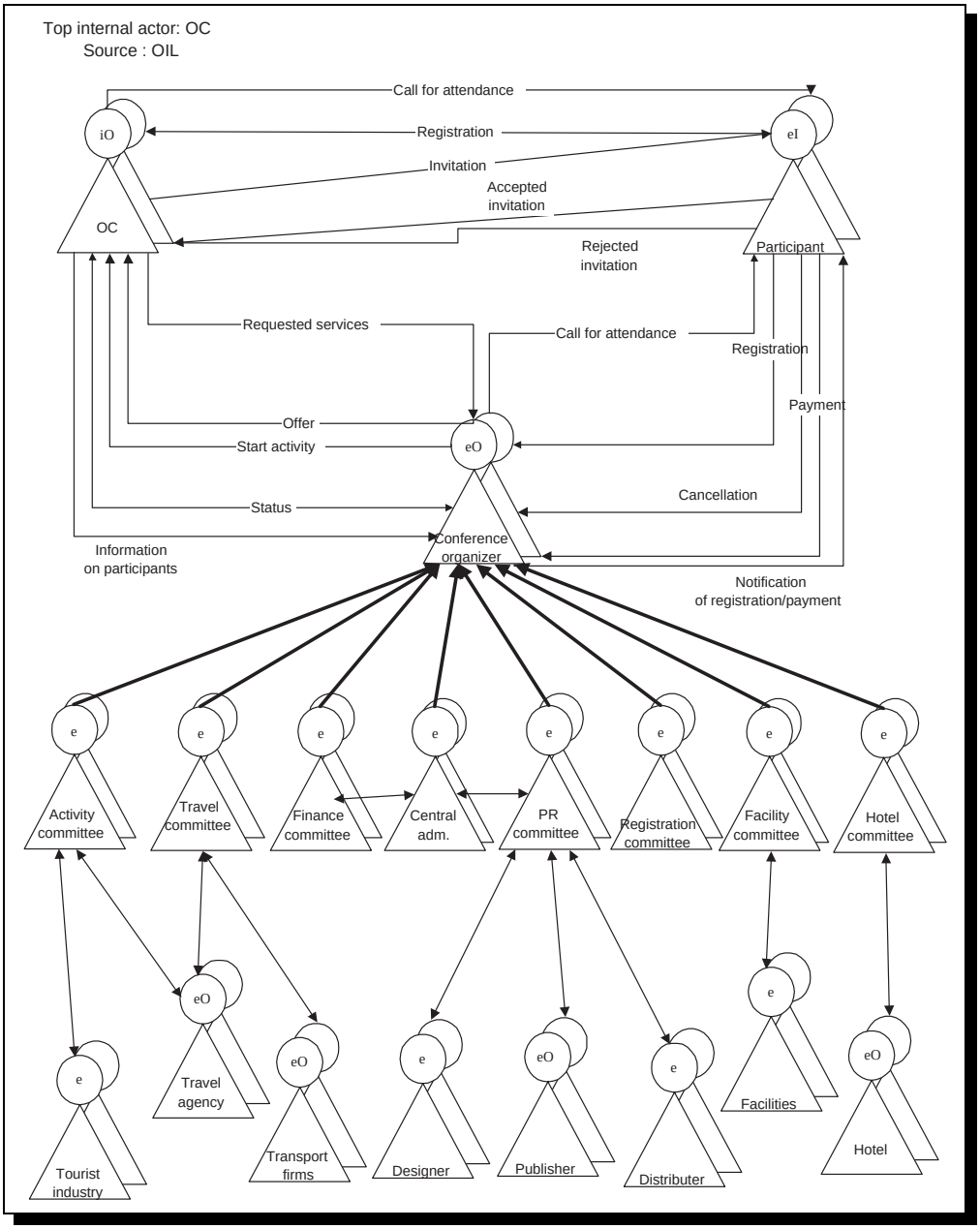

Figure A.25: Actor model describing interactions in connection with conference organization

Roles

- R3 Publisher: External organizational.
- R4 Conference organizer: External organizational.
- R6 Organizing committee: Internal organizational.
- R8 Activity committee: External.
- R9 Travel committee: External.
- R10 Finance committee: External.
- R11 Central administration: External.
- R12 PR committee: External.
- R13 Registration committee: External.
- R14 Facilities committee: External.
- R15 Hotel committee: External.
- R16 Tourist industry: External.
- R17 Travel agency: External organizational.
- R18 Transport firm: External organizational.
- R19 Designer: External.
- R21 Distributer: External.
- R22 Facilities: External.
- R23 Hotel: External organizational.
- R152 Participant: External individual.

Relationship	From	Role	Behalf	To	Comment
Part-of		R8		R4	
Part-of		R9		R4	
Part-of		R10		R4	
Part-of		R11		R4	
Part-of		R12		R4	
Part-of		R13		R4	
Part-of		R14		R4	
Part-of		R15		R4	

Flows

- f1 Call for attendance.
- f2 Registration.
- f3 Invitation: Invitation to participate in the conference.
- f4 Accepted invitation.
- f5 Rejected invitation.
- f6 Requested services: The organizing committee requests services from the conference organizer.
- f7 Offer.
- f8 Start activity: Reminder sent from the conference organizer to the organizing committee to start a given activity.
- f9 Status.
- f10 Information on participants.
- f11 Call for attendance.
- f12 Registration.
- f13 Cancelation: Cancelation of registration.
- f14 Payment.

- f15 Notification of registration/payment: An acknowledgment on that the registration/payment is received.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R6			R152	
f2		R152			R6	
f3		R6			R152	
f4		R152			R6	
f5		R152			R6	
f6		R6			R4	
f7		R4			R6	
f8		R4			R6	
f9		R4			R6	
f10		R6			R4	
f11		R4			R152	
f12		R152			R4	
f13		R152			R4	
f14		R152			R4	
f15		R4			R152	
f16		R10			R11	
f17		R11			R10	
f18		R11			R12	
f19		R12			R11	
f20		R8			R16	
f21		R16			R8	
f22		R8			R17	
f23		R17			R8	
f24		R9			R17	
f25		R17			R9	
f26		R9			R18	
f27		R18			R9	
f28		R12			R19	
f29		R19			R12	
f30		R12			R3	
f31		R3			R12	
f32		R12			R21	
f33		R21			R12	
f34		R14			R22	
f35		R22			R14	
f36		R15			R23	
f37		R23			R15	

Overall reconciled process-model :

Main source: ISDO. Top internal actor. Conference OC. The diagram is given in Figure A.26 and is based on the previous diagrams. It is not indicated who will support the different processes.

Roles

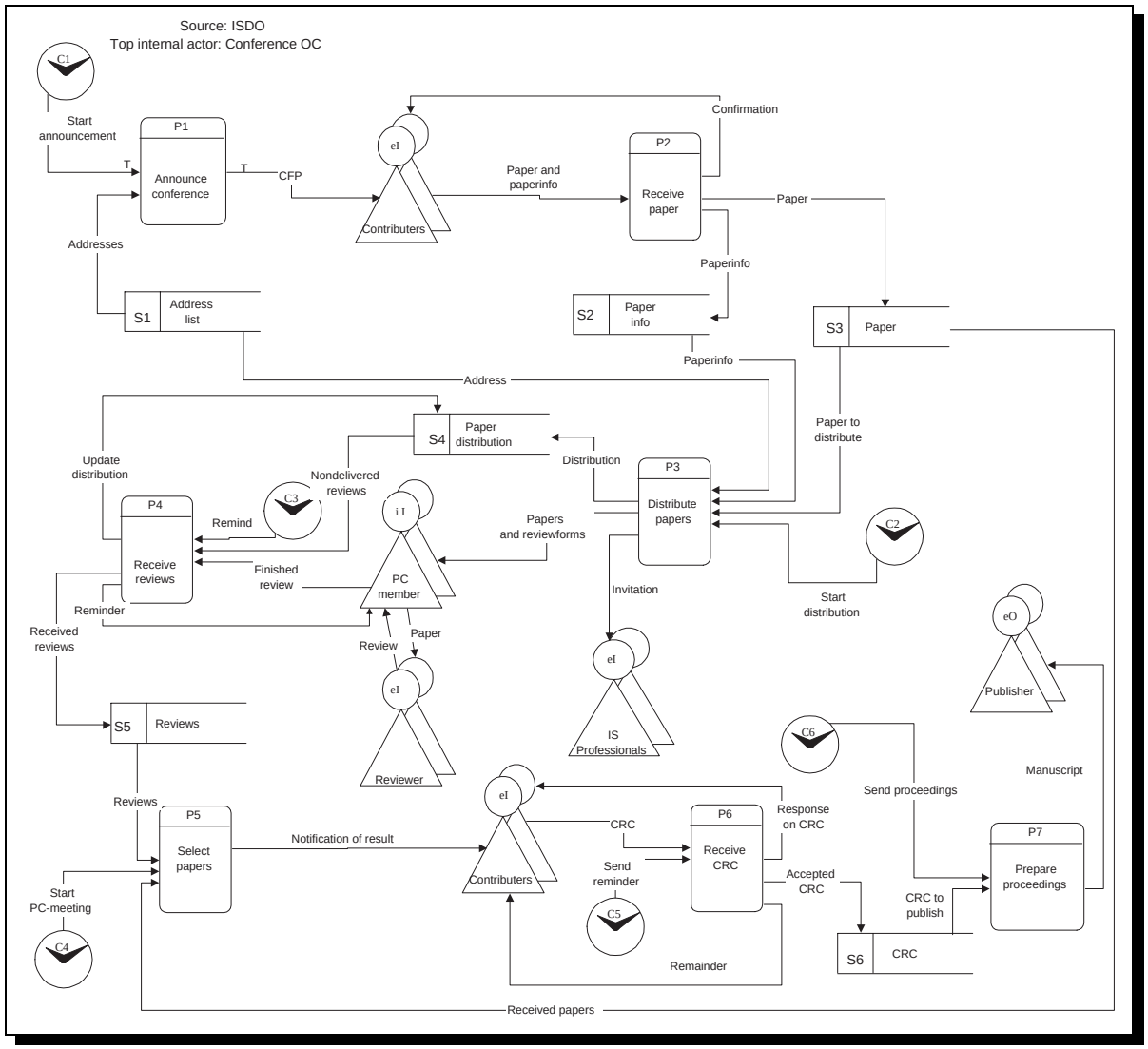

Figure A.26: Overall reconciled PPM

- R5 Publisher: External organizational. Publisher of e.g. proceedings.
- R108 PC member: External individual.
- R149 Contributer: External individual. Someone potentially contributing a paper to the conference.
- R150 IS professional: External individual. A prospective additional member of the program committee.
- R151 Reviewer: External individual. A reviewer not part of the program committee.

Stores

- S1 Address list: Address information of all stakeholders.
- S2 Paper info: Information about submitted papers.
- S3 Paper: The physical papers.
- S4 Paper distribution: Overview our who is to review which papers.
- S5 Reviews.
- S6 CRC: Camera ready copies of the accepted papers.

Timers

- C1: To trigger the announcement of the conference.
- C2: To trigger the distribution of received papers to reviewers.
- C3: To trigger the sending of reminders to PC-members that have not given in their reviews.
- C4: To trigger the selection of papers on the PC-meeting.
- C5: To trigger the sending of reminders on CRCs.
- C6: To trigger the creation of the manuscript of the proceedings.

Processes

- P1 Announce conference.
- P2 Receive papers.
- P3 Distribute papers.
- P4 Receive reviews.
- P5 Select papers.
- P6 Receive CRC.
- P7 Prepare proceedings.

Flows

- f1 Start announcement.
- f2 Addresses.
- f3 CFP.
- f4 Paper and paperinfo.
- f5 Confirmation.
- f6 Paper.
- f7 Paperinfo.
- f8 Start distribution.
- f9 Paper to distribute.
- f10 Paperinfo.
- f11 Address.
- f12 Distribution.
- f13 Papers and reviewform.
- f14 Invitation.
- f15 Finished review.
- f16 Nondelivered reviews
- f17 Remind.
- f18 Update distribution.
- f19 Received reviews.
- f20 Reminder.
- f21 Start PC-meeting
- f22 Reviews.
- f23 Received papers.
- f24 Notification of result.
- f25 CRC.
- f26 Send reminder.
- f27 Response on CRC.
- f28 Accepted CRC.
- f29 Reminder.
- f30 Send proceedings.
- f31 CRC to publish.

- f32 Manuscript.
- f33 Paper.
- f34 Review.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	C1			P1		
f2	S2			P1		
f3	P1				R149	
f4		R149		P2		
f5	P2				R149	
f6	P2			S3		
f7	P2			S2		
f8	C2			P3		
f9	S3			P3		
f10	S2			P3		
f11	S1			P3		
f12	P3			S4		
f13	P3				R108	
f14	P3				R150	
f15		R108		P4		
f16	S4			P4		
f17	C3			P4		
f18	P4			S4		
f19	P4			S5		
f20	P4				R108	
f21	C4			P5		
f22	S5			P5		
f23	S3			P5		
f24	P5				R149	
f25		R149		P6		
f26	C5			P6		
f27	P6				R149	
f28	P6			S6		
f29	P6				R149	
f30	C6			P7		
f31	S6			P7		
f32	P7				R5	
f33		R108			R151	
f34		R151			R108	

Speech acts on the interaction between a conference organizer and an IS-professional :

The diagram is given in Figure A.27. Main source: JK. The individual speech acts are further described below.

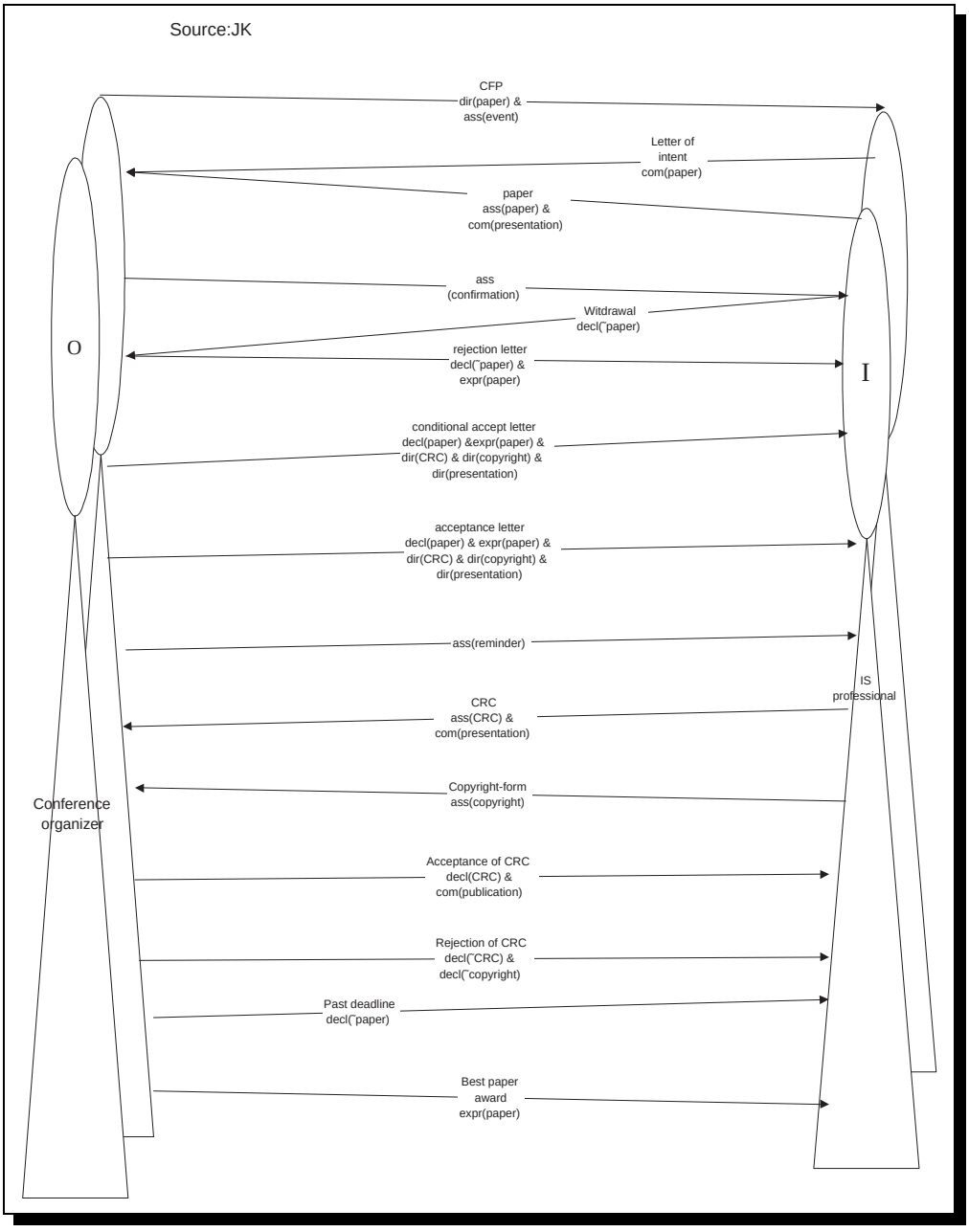

Figure A.27: Speech acts in the paper-process

Roles

- R6 Organizing committee. Internal organizational
- R150 IS-professional. External individual.

Flows: The presentation is based on the sending of items. The illocutionary acts are presented as a triplet <illocutionary point, propositional content, dominant claim> and implied rules are listed after the illocutionary point if any, followed by a comment.

- f1 CFP: Call for paper
  - < dir, paper, truth >

    ```
    If before CFP-deadline
    it is permitted for IS-professionals to issue a paper to the conference.

    If issuing a paper to the conference
    it is recommended for the issuer to write
    within the areas of interest indicated in the CFP.
    ```

  - < ass, conference, truth >.
- f2 Letter of intent: Indicates that the IS professional is going to issue a paper to the conference.
  - <com, paper, sincerity >

    ```
    when letterofintent(Person,Conference)
    if before CFP-deadline
    and Paper written by Person  has not been issued to Conference
    it is recommended for Person to issue a Paper to Conference within the
    CFP-deadline
    ```

- f3 Paper:
  - < ass, paper, justice >
  - < com, presentation, justice >

    ```
    when issuepaper(Authors,Paper,Conference)
    if Paper accepted
    it is obligatory for at least one of the Authors to
    attend the Conference and present the Paper there.

    when issuepaper(Paper,Conference)
    and not withdraw(Paper) after issuepaper(Paper,Conference)
    it is obligatory for the Conference organizers of Conference
    to see to that Paper is reviewed.
    ```

- f4 Confirmation: Confirming the reception of paper.
  - <ass, confirmation, truth >
- f5 Witdrawal: Withdrawal of submitted paper.
  - < decl , not paper, justice >

    ```
    When withdrawal(Paper)
    if Paper distributed to Reviewer
    it is obligatory to notice the reviewer of the changed situation.
    and update overview of  reviews to be expected.
    ```

- f6 Rejection letter.
  - <decl, not paper, justice>
  - <expr, paper, sincerity>

- f7 Conditional accept letter.
  - < decl, paper, justice >
  - < expr, paper, sincerity > (the actual reviews)
  - < dir, CRC, justice >

    ```
    If the paper is to be printed in the proceedings
    it is obligatory for the Authors to make and return a
    CRC following the style-guide
    and taking the review-comments into account within the CRC-deadline.
    ```
  - < dir, copyright, justice >

    ```
    if the paper is to be printed in the proceedings
    it is obligatory for the authors to pass over copyright to the publishers.
    ```
  - < dir, presentation, justice >

    ```
    it is obligatory for at least one of the authors to attend the
    conference and present the paper.
    ```
- f8 Acceptance letter.
  - < decl, paper, justice >
  - < expr, paper, sincerity >
  - < dir, CRC, justice >

    ```
    If the paper is to be printed in the proceedings
    it is obligatory for the Authors to make and return a
    CRC following the style-guide within the CRC-deadline.

    If the paper is to be printed in the proceedings
    it is recommended for the Authors to make and return a
    CRC taking the review-comments into account
    within the CRC-deadline.
    ```
  - < dir, copyright, justice >

    ```
    if the paper is to be printed in the proceedings
    it is obligatory for the authors to pass over copyright to the publishers.
    ```
  - < dir, presentation, justice >

    ```
    it is obligatory for at least one of the authors to attend the
    conference and present the paper.
    ```
- f9 Reminder: Reminder for the issuing of a CRC.
  - <ass, reminder, truth >
- f10 CRC.
  - <ass, CRC, justice >
  - < com, presentation, justice >
- f11 Copyright-form: Signed copyright-form.
  - <ass, copyright, justice >
- f12 Acceptance of CRC.
  - < decl, paper, justice >
  - < com, publication, justice >

    ```
    When CRC is accepted for conference
    and the copyrightform is returned
    it is obligatory for the organizers of Conference to publish CRC
    ```

- f13 Rejection of CRC.
  - < decl, not paper, justice >
  - <decl, not copyright, justice >
- f14 Past deadline: CRC is rejected since it is past the deadline.
  - < decl, not paper, justice >
- f15 Best paper award.
  - < expr, paper, sincerity >

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R6			R150	
f2		R150			R6	
f3		R150			R6	
f4		R6			R150	
f5		R150			R6	
f6		R6			R150	
f7		R6			R150	
f8		R6			R150	
f9		R6			R150	
f10		R150			R6	
f11		R150			R6	
f12		R6			R150	
f13		R6			R150	
f14		R6			R150	
f15		R6			R150	

Model of the improved situation

At this stage it was decided to concentrate only on the paper-process due to time limitations. We have in this section included both the models of this with and without CIS-support. The indication of CIS-support is not given in the diagrams, but rather in the text where it is specified which process is supported by which actor in version 1 and 2 of the system. The process-model was started based on Figure A.26.

We have also indicated the data-model for the situation and the rules and rule-relationships. The overall data model is given first, the the process-models indicating in the text which parts will be supported by the C4-conference-program in version 1 and 2 respectively below. For one of the processes, a more detailed specification of rules and PLDs are given. Finally an actor-model of the C4-actor is presented. We will only indicate the source of rules here, and not the graphical models, since they at this stage did not have a distinct source.

Developed ONER model :

The diagram is given in Figure A.28.

Types

- T1 Name: Composite(Att2 Firstname:String(n), Att3 Initials:String(n)*, Att4 Lastname:String(n))
- T2 Chair: Person (E1, description given below).
- T3 Editors: Set of Conference organizers (E4).
- T4 Statelist: Set of String(n).

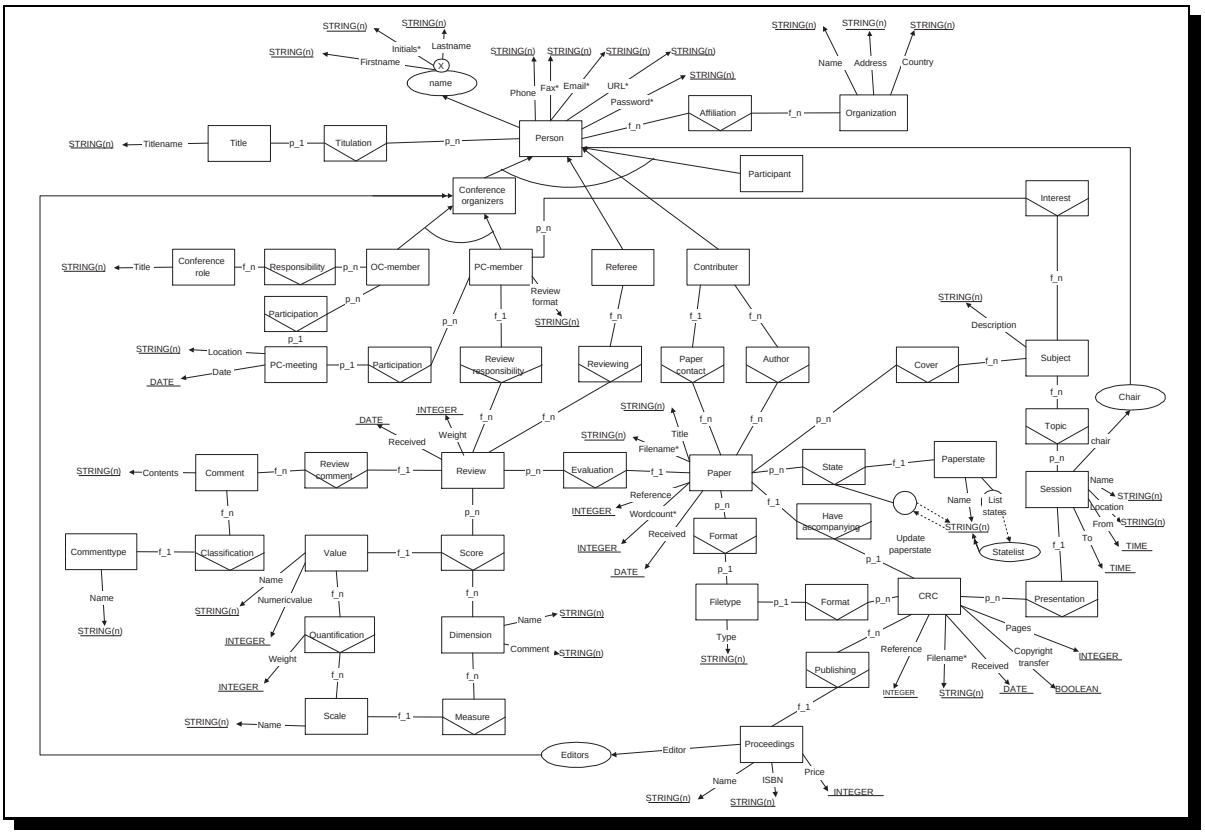

Figure A.28: ONER-model for the conference system

Entities

- E1 Person.
  Attributes
  - Att1 Name:T1.
  - Att5 Phone:String(n).
  - Att6:Fax:String(n). Optional.
  - Att7:Email:String(n). Optional.
  - Att8:URL:String(n) Optional. WWW-identifier to support linking to persons with own WWW home-page from the conference web-pages.
  - Att9:Password:String(n). Optional. Mandatory for PC-members to support the issuing of reviews by WWW.
- E2 Organization.
  Attributes
  - Att10 Name:String(n). Name of organization.
  - Att11 Address:String(n). Postal address of organization.
  - Att12 Country:String(n).
- E3 Title.
  Attributes
  - Att13 Titlename:String(n). Values: Prof. Dr, Prof, Dr, Mr, Ms.
- E4 Conference organizers: Superclass E1.
- E5 OC-member: Superclass E4.

- E6 PC-member: Superclass E4.
  Attributes
  - Att48 Reviewformat:String(n). Values: Paper/Electronical. Indicates the preferred format of papers by the reviewers.
- E7 Referee: Referee of paper. Superclass E1.
- E8 Contributer: Author of paper. Superclass E1.
- E9 Participant: Participant at the conference. Superclass E1.
- E10 Conference role: A role being institutionalized by a conference organization.
  Attributes
  - Att14 Title:String(n). Values: General chair, Program chair, Organizing chair, Program Coordinator.
- E11 PC-meeting: Meeting by the program committee.
  Attributes
  - Att15 Location:String(n). The location of the PC-meeting.
  - Att16 Date:date. The date of the PC-meeting.
- E12 Review.
  Attributes
  - Att25 Date:Date. Date of the reception of the review.
  - Att26 Weight:Integer. The weight for a given review. Default = 1.
- E13 Comment: A textual comment in a review.
  Attributes
  - Att17 Contents:String(n). The actual comment.
- E14 Commenttype: Type of comment requested in a review.
  Attributes
  - Att18 Name:String(n). Values: Main contributions, Positive aspects, Negative aspects, Comments to the author.
- E15 Value. A value that could be given a review according to a given dimension.
  Attributes
  - Att19 Name:String(n).
  - Att20 Numericvalue:Integer.
- E16 Scale. Scale along which certain dimensions where measured.
  Attributes
  - Att22 Name:String(n). Values: OneToTen (1-10), NoneToaLot (None, Some, A lot), LittleToHigh (Little, Medium, High).
- E17 Dimension. Evaluation dimension, such as originality, significance etc.
  Attributes
  - Att23 Name:String(n) Values: Originality, Significance of Topic, Technical quality, Relevance to conference, Presentation, Overall, Rewriting needed, Reviewer expertise.
- E18 Paper.
  Attributes
  - Att27 Title:String(n). Title of paper.
  - Att28 Filename:String(n). Optional. Filename of paper if it had been submitted electronically.
  - Att29 Reference:Integer. A unique, meaningful reference.
  - Att30 Wordcount:Integer. Optional.
  - Att31 Received:Date. The date the paper was received.

- E19 Subject: Topic of a paper/session.
  Attributes
  - Att47 Description:String(n). Keyword or phrase indicating a subject area.
- E20 Session: Session of the conference in which two or more papers are combined.
  Attributes
  - Att42 To:Time. End-time of session.
  - Att43 From:Time. Start-time of session.
  - Att44 Location:String(n). Where the session is to take place.
  - Att45 Name:String(n). Name of session.
  - Att46 Chair:Chair. Chairman of the session.
- E21 Paperstate: Possible state of a paper.
  Methods: M1.
  Attributes
  - Name:String(n). Values: Received, Distributed, In discussion, Accepted, Conditionally accepted, Rejected, CRC, Published.
- E22 CRC: CRC of an accepted paper.
  Attributes
  - Att37 Reference: Integer. Numeric reference to paper.
  - Att38 Filename:String(n). Optional.
  - Att39 Received:Date.
  - Att40 Copyright transfer:Boolean. Indicate if the copyright is transferred on the paper or not.
  - Att41 Pages: Number of pages of CRC.
- E23 Filetype: Format of paper and CRC.
  Attributes
  - Att49 Type:String(n). Values: Physical, Postscript, RTF.
- E24 Proceedings: The proceedings including the CRCs.
  Attributes
  - Att32 Editors:T3.
  - Att33 Name:String(n).
  - Att34 ISBN:String(n).
  - Att35 Price:Integer.

Relationships

- Rel1 Titulation.
  Involves
  - E1 pn
  - E3 p1
- Rel2 Affiliation.
  Involves
  - E1 fn
  - E2 fn
- Rel3 Responsibility.
  Involves
  - E5 pn
  - E10 fn
- Rel4 Participation.
  Involves

  - E5 pn
  - E11 p1
- Rel5 Participation.
  Involves
  - E6 pn
  - E11 p1
- Rel6 Review responsibility.
  Involves
  - E6 f1
  - E12 fn
- Rel7 Reviewing.
  Involves
  - E7 fn
  - E12 fn
- Rel8 Paper contact.
  Involves
  - E8 f1
  - E18 fn
- Rel9 Author.
  Involves
  - E8 fn
  - E18 fn
- Rel10 Evaluation.
  Involves
  - E18 f1
  - E12 pn
- Rel11 Review comments.
  Involves
  - E12 f1
  - E13 fn
- Rel12 Classification.
  Involves
  - E13 fn
  - E14 f1
- Rel13 Quantification.
  Attributes:
  Att21 Weight:Integer, Default = 1.
  Involves
  - E15 fn
  - E16 fn
- Rel14 Score.
  Involves
  - E12 pn
  - E15 f1
  - E17 fn
- Rel15 Measure.
  Involves

  - E17 fn
  - E16 f1
- Rel16 Format.
  Involves
  - E18 pn
  - E23 p1
- Rel17 Format.
  Involves
  - E22 pn
  - E23 p1
- Rel18 Publishing.
  Involves
  - E22 fn
  - E24 f1
- Rel19 Have accompanying.
  Involves
  - E18 f1
  - E22 p1
- Rel20 State.
  Methods: M2
  Involves
  - E18 pn
  - E21 f1
- Rel21 Cover.
  Involves
  - E18 pn
  - E19 fn
- Rel22 Presentation.
  Involves
  - E22 pn
  - E20 f1
- Rel23 Interest.
  Involves
  - E4 pn
  - E19 fn
- Rel24 Topic.
  Involves
  - E19 fn
  - E20 pn

Methods

- M1 List states:T4.
- M2 Update paperstate(String(n)):String(n).

Process P1: Creation and distribution of CFP decomposed :

The diagram is given in Figure A.29. In due time the CFP is created, and is partly distributed in a multi-casting fashion to the research community and the PC-members, but also as results of specific requests. It is also possible for people to retrieve the CFP by WWW.

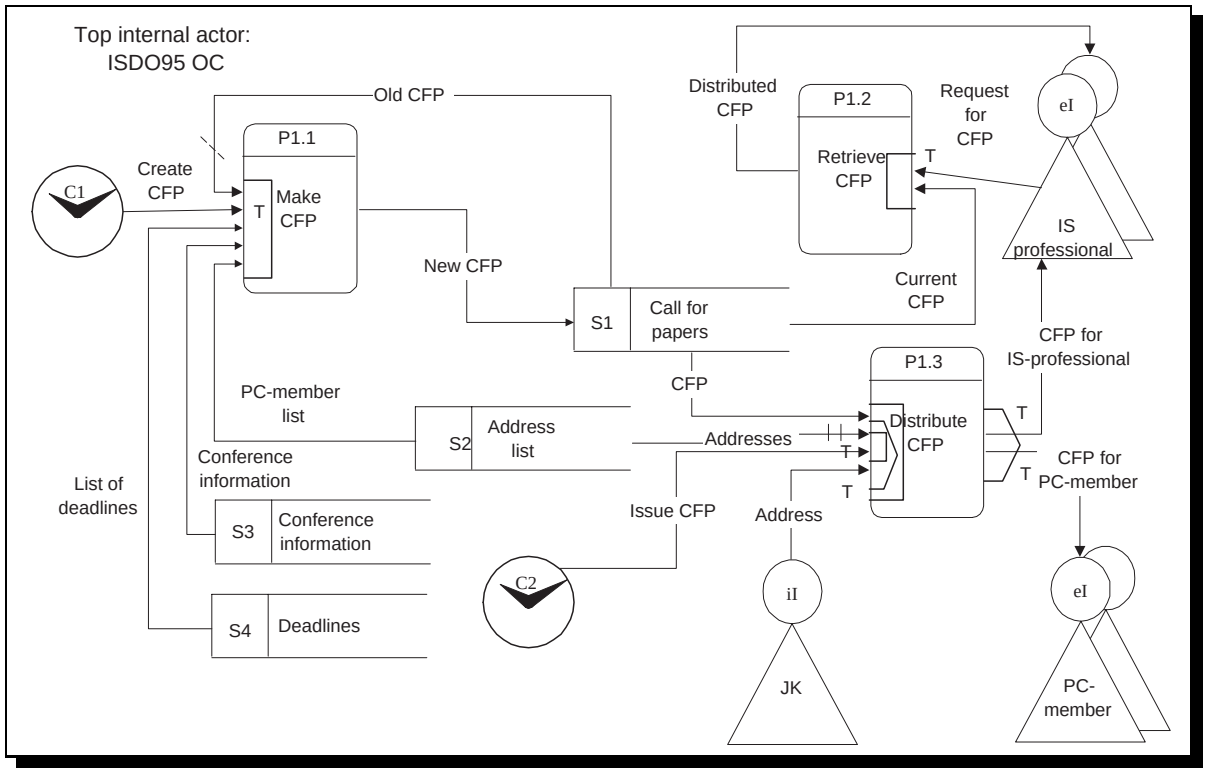

Figure A.29: PPM describing new creation and distribution of CFP

Actors

- A22 ISDO95: External organizational.
- A101 AS: Internal individual.
- A102 JK: Internal individual.
- A255 C4web: Internal applicative. The WWW-part of C4.
- A256 C4shell: Internal applicative. The Unix-shell part of C4.

Roles

- R108 PC-member: External individual.
- R112 Program coordinator: External individual.
- R146 Program chair: External individual.
- R150 IS professional: External individual.

Stores

- S1 Call for papers.
- S2 Address list.
- S3 Conference information: General information about the conference, including topic of conference.
- S4 Deadlines: Deadlines for CFP, CRC etc.

Timers

- C1: Indicate the appropriate time for creating the CFP.
- C2: Indicate the appropriate time(s) for issuing the CFP on a large scale.

Processes

- P1.1 Make CFP: Create the physical and electronical version of the call for paper.
- P1.2 Retrieve CFP: Retrieval of electronic CFP using WWW.

- P1.3 Distribute CFP.

Flows

- f1 Create CFP: Signal.
- f2 Old CFP: Previous version of CFP if any.
- f3 List of deadlines.
- f4 Conference information.
- f5 PC-member list.
- f6 New CFP.
- f7 CFP.
- f8 Addresses: Addresses used for bulk distribution.
- f9 Issue CFP: Signal.
- f10 Address: Address to send single CFP to.
- f11 CFP for IS-professional.
- f12 CFP for PC-member.
- f13 Current CFP.
- f14 Request for CFP.
- f15 Distributed CFP.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	C1			P1.1		
f2	S1			P1.1		
f3	S4			P1.1.		
f4	S3			P1.1		
f5	S2			P1.1		
f6	P1.1			S1		
f7	S1			P1.3		
f8	S2			P1.3		
f9	C2			P1.3		
f10	A102	R112	A22	P1.3		
f11	P1.3				R150	
f12	P1.3				R108	
f13	S1			P1.2		
f14		R150		P1.2		
f15	P1.2				R150	

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A101	R146	A22	P1.1				
A255			P1.2			Potential	
A256			P1.3			Potential	

Version 2

Similar to version 1, but the status is no longer potential.

Process P2: Reception of papers :

The diagram is given in Figure A.30. Papers are either issued by post or by ftp. Information on papers are given by post, email, or our WWW after that the paper is issued by ftp. The information is controlled to check that it contains sufficient information, before the paper is registered. A confirmation of the reception of the paper is issued if everything is OK. If a paper sent by ftp can not be printed, the contributer is asked to resend the paper.

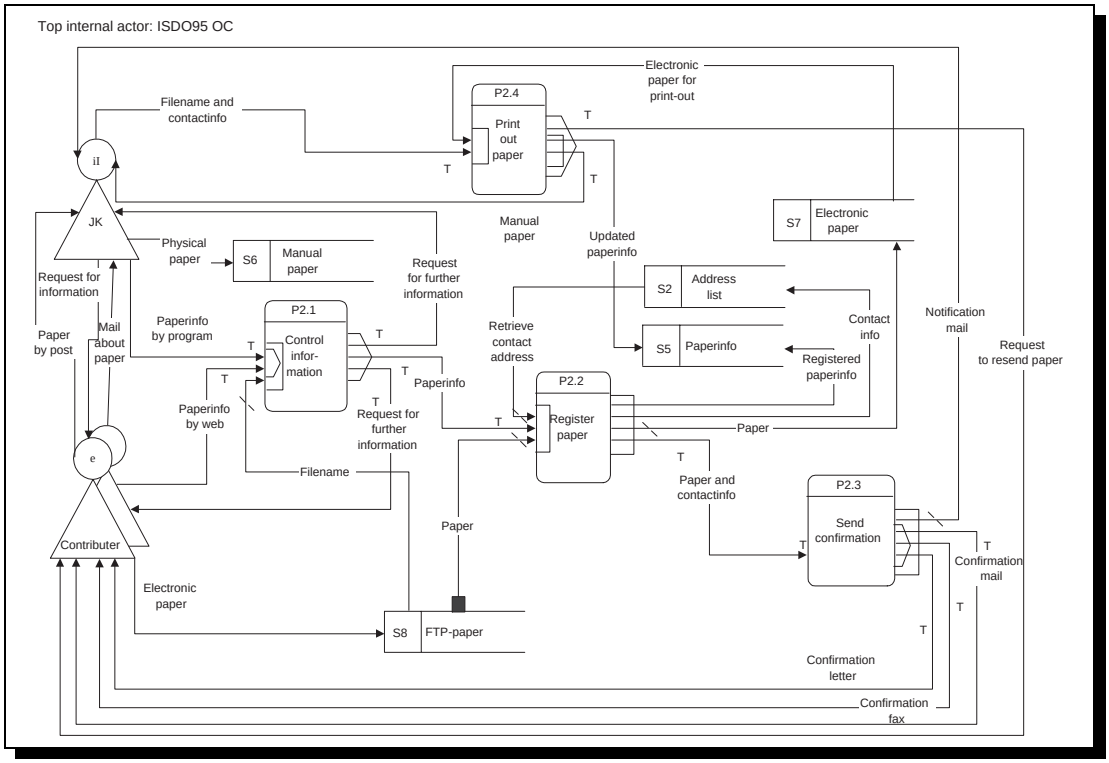

Figure A.30: PPM describing new reception of papers

Actors

- A22 ISDO95: External organizational.
- A102 JK: Internal individual.
- A230 C4: Internal applicative.

Roles

- R112 Program coordinator: External individual.
- R149 Contributer: External.

Stores

- S2 Address list.
- S5 Paperinfo.
- S6 Manual paper: Store of physical papers.
- S7 Electronic papers: Directory for the storage of papers in postscript format.
- S8 FTP-paper: Ftp-directory. Papers that are issued electronically are temporarily stored here, but are removed as soon as possible.

Processes

- P2.1 Control information: Controls that the information issued about papers are sufficiently detailed.

- P2.2 Register paper: Register details about papers and transfers electronic papers away from the ftp-directory.
- P2.3 Send confirmation: Send a confirmation that the paper is received and registered.
- P2.4 Print out paper: Take paper copy of electronic paper. If it is not possible to print the paper, ask for the paper to be resent.

Flows

- f1 Mail about paper: Email giving information about a submitted paper.
- f2 Request for information: Requesting further information about the contact person and authors of a paper.
- f3 Paper by post: Paper submitted in five copies by postal mail.
- f4 Paperinfo by program.
- f5 Paperinfo by web.
- f6 Filename: Name of file for electronic paper being submitted by ftp.
- f7 Request for further information.
- f8 Paperinfo: Sufficient paperinfo sent to registration.
- f9 Request for further information.
- f10 Retrieve contact address: If already exist.
- f11 Paper.
- f12 Registered paperinfo.
- f13 Contact info.
- f14 Paper.
- f15 Paper and contactinfo.
- f16 Notification mail: Sent in case the information is entered by the contributer.
- f17 Confirmation mail.
- f18 Confirmation fax: Sent if the contributer has not given an email address.
- f19 Confirmation letter: Sent if the contributer has neither given an email address or a fax-number.
- f20 Electronic paper: Paper submitted by ftp.
- f21 Physical paper.
- f22 Filename and contactinfo.
- f23 Electronic paper.
- f24 Request to resend paper.
- f25 Updated paperinfo.
- f26 Manual paper.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R149		A102	R112	A22
f2	A102	R112	A22		R149	
f3		R149		A102	R112	A22
f4	A102	R112	A22	P2.1		
f5		R149		P2.1		
f6	S8			P2.1		
f7	P2.1			A102	R122	A22
f8	P2.1			P2.2		
f9	P2.1				R149	

continued on next page

continued from previous page						
	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f10	S2			P2.2		
f11	S8			P2.2		
f12	P2.2			S5		
f13	P2.2			S2		
f14	P2.2			S7		
f15	P2.2			P2.3		
f16	P2.3			A102	R122	A22
f17	P2.3				R149	
f18	P2.3				R149	
f19	P2.3				R149	
f20		R149		S8		
f21	A102	R112	A22	S6		
f22	A102	R112	A22	P2.4		
f23	S7			P2.4		
f24	P2.4				R149	
f25	P2.4			S5		
f26	P2.4			A102	R112	A22

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A230			P2.1			Potential	
A230			P2.2			Potential	
A102	R112	A22	P2.3				
A102	R112	A22	P2.4				

Version 2

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A230			P2.1				
A230			P2.2				
A230			P2.3			Potential	
A102	R112	A22	P2.4				

Process P3: Distribution of papers to reviewers :

The diagram is given in Figure A.31. The model is similar to what was described in the model of the perceived situation, with some additions because of the possibility of electronic papers and reviews.

Actors

- A22 ISDO95: External organizational.
- A38 ISDO OC: Internal organizational. Organizing committee of ISDO95.
- A101 AS : Internal individual.

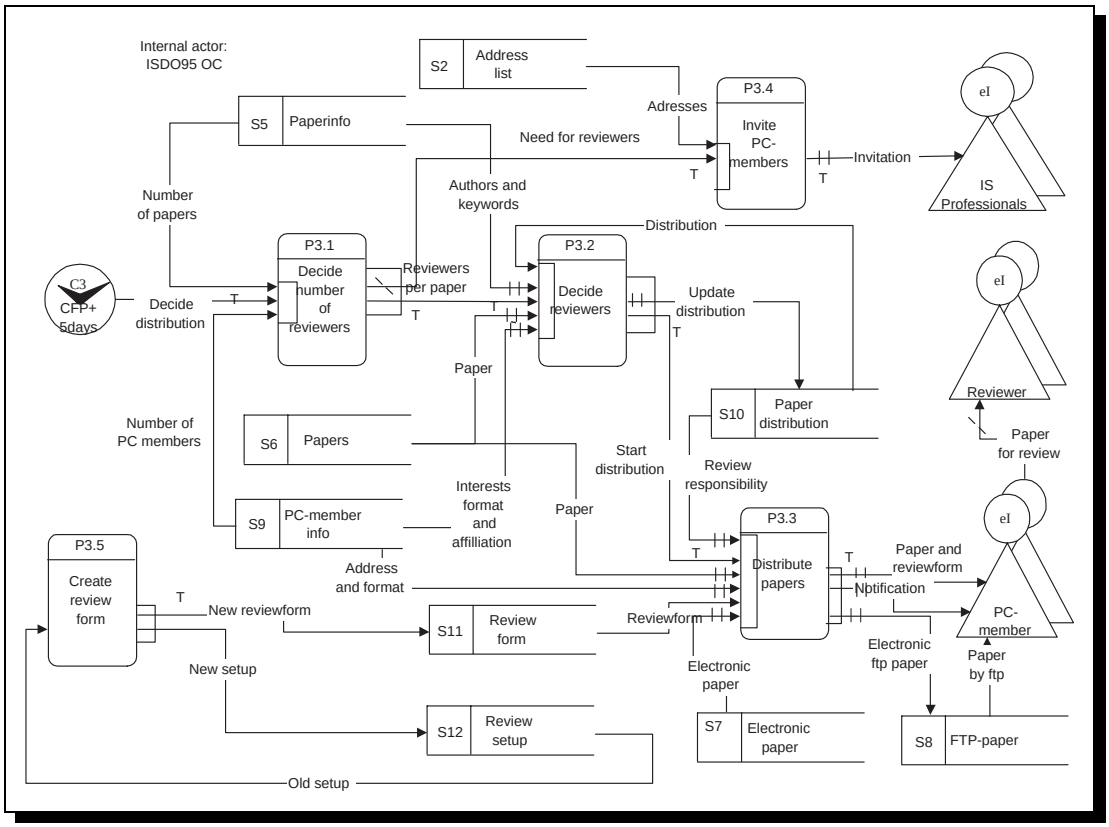

Figure A.31: PPM describing new distribution of paper

- A102 JK : Internal individual.
- A230 C4: Internal applicative.

Roles

- R108 PC-member. External individual.
- R112 Program coordinator: Internal individual.
- R146 Program chair: Internal individual.
- R150 IS-professional: External individual.
- R151 Reviewer: External individual.

Stores

- S5 Paperinfo.
- S6 Manual paper: Store of physical papers.
- S7 Electronic papers: Directory for the storage of papers in postscript format.
- S8 FTP-paper: Ftp-directory where papers to be reviewed can be down-loaded directly by the PC-member.
- S9 PC-member info.
- S10 Paper distribution: Indicate which PC-members that are responsible for reviewing which papers.
- S11 Review form.
- S12 Review setup.

Timers

- C3 CFP deadline + 5days: Wait some time after the CFP-deadline before performing the

distribution, since one often receives papers some days after the deadline.

Processes

- P3.1 Decide number of reviewers: Decide how many should review each paper. If many papers have been received, one might need to invite further PC-members.
- P3.2 Decide reviewers: Decide which papers that should be reviewed by which PC-members.
- P3.3 Distribute papers: and review forms.
- P3.4 Invite PC-members: Invite PC-members to the program committee.
- P3.5 Create reviewform: Include the decision of comment-types and evaluation-dimensions and their scales to be used in the review-forms.

Flows

- f1 Decide distribution: Signal.
- f2 Number of papers.
- f3 Number of PC-members.
- f4 Need for reviewers.
- f5 Reviewers per paper.
- f6 Distribution: Existing distribution of papers if any.
- f7 Authors and keywords.
- f8 Paper.
- f9 Interests, formats, and affiliation: Format indicate if the PC-member would like to use the electronic facilities.
- f10 Update distribution.
- f11 Start distribution.
- f12 Addresses: Addresses of potential PC-members.
- f13 Invitation: Invitation to participate in the program committee.
- f14 Review responsibility.
- f15 Paper: Copy of physical paper.
- f16 Address and format.
- f17 Reviewform.
- f18 Electronic paper.
- f19 Paper and reviewform.
- f20 Notification: Sent to those that are to down-load and print out the papers to review themselves.
- f21 Electronic ftp paper.
- f22 Paper by ftp.
- f23 Paper for review.
- f24 Old setup: Previous review setup if any.
- f25 New reviewform.
- f26 New setup.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	C3			P3.1		
f2	S5			P3.1		
f3	S9			P3.1		
f4	P3.1			P3.4		
f5	P3.1			P3.2		

continued on next page

continued from previous page

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f6	S10			P3.2		
f7	S5			P3.2		
f8	S6			P3.2		
f9	S9			P3.2		
f10	P3.2			S10		
f11	P3.2			P3.3		
f12	S2			P3.4		
f13	P3.4				R150	
f14	S10			P3.3		
f15	S6			P3.3		
f16	S9			P3.3		
f17	S11			P3.3		
f18	S7			P3.3		
f19	P3.3				R108	
f20	P3.3				R108	
f21	P3.3			S8		
f22	S8				R108	
f23		R108			R151	
f24	S12			P3.5		
f25	P3.5			S11		
f26	P3.5			S12		

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A38		A22	P3.1				
A38		A22	P3.2				
A102	R112	A22	P3.3				
A101	R146	A22	P3.4				
A38		A22	P3.5				

Version 2

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A38		A22	P3.1				
A38		A22	P3.2				
A230			P3.3			Potential	
A101	R146	A22	P3.4				
A38		A22	P3.5				

Process P4: Receive and remind reviews :

This is divided into two diagrams, one for the reception of reviews and one for the reminding of reviews.

The diagram is given in Figure A.32.

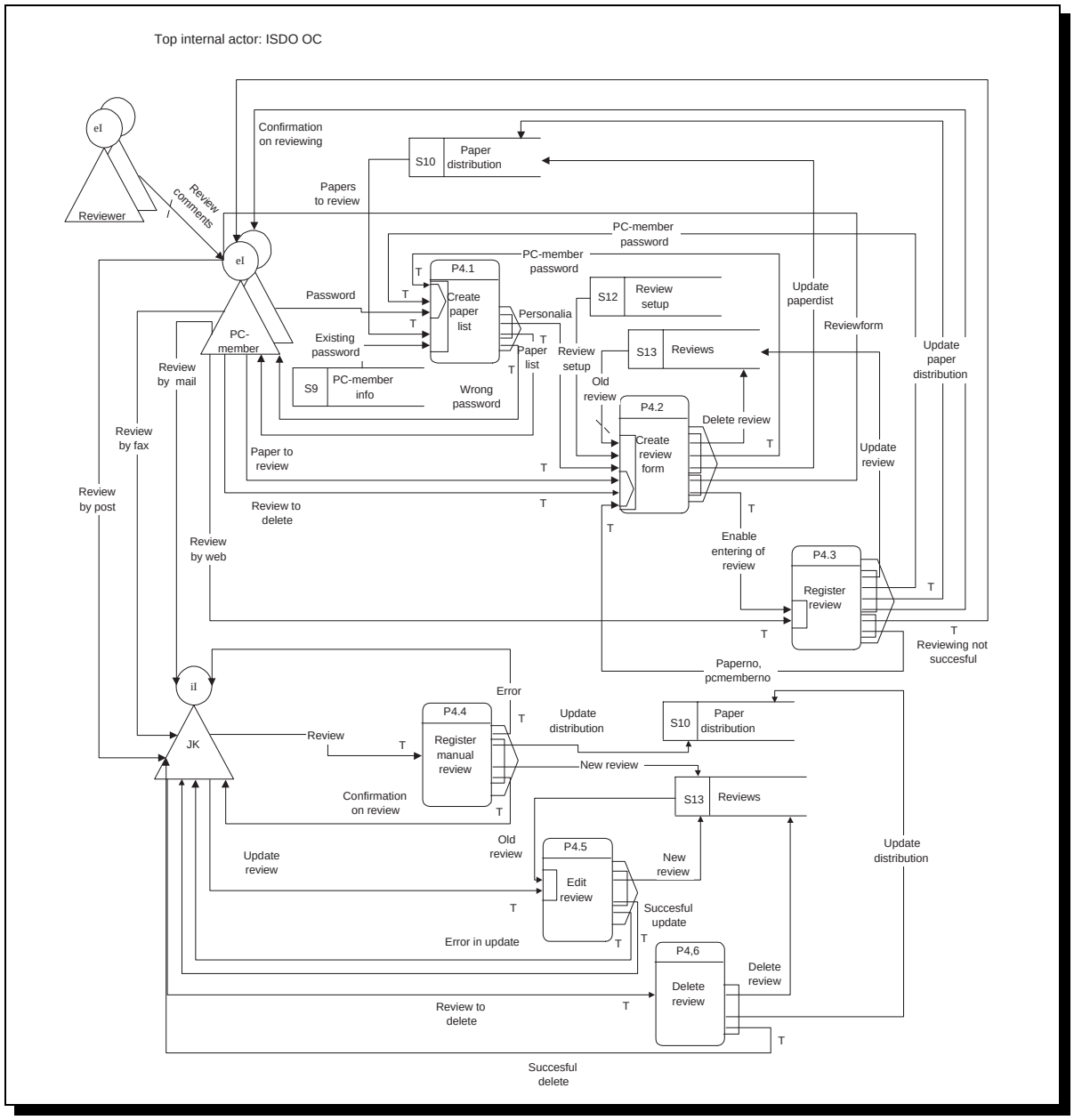

Figure A.32: PPM describing new reception of reviews

Actors

- A22 ISDO95: External organizational.
- A102 JK: Internal Individual.
- A255 C4web: Internal applicative. The WWW-part of the C4.
- A257 C4motif: Internal applicative. The Motif part of C4.

Roles

- R108 PC-member: External individual.
- R112 Program coordinator: Internal individual.
- R151 Reviewer: External individual.

Stores

- S9 PC-member info.
- S10 Paper distribution: Indicate which PC-members that are responsible for reviewing which papers.
- S12 Review setup.
- S13 Reviews: Evaluations of papers.

Processes

- P4.1 Create paper list: Based on the password given, a personal list over papers to be reviewed, and papers already reviewed by the reviewer is created.
- P4.2 Create reviewform: If requested, the review form for the review of a specific paper is created dynamically based on the review setup. If edit review is chosen, the already submitted review is also retrieved. A submitted review can also be deleted.
- P4.3 Register review: A review is registered.
- P4.4 Register manual review: A review received by post, fax, or email is registered.
- P4.5 Edit review.
- P4.6 Delete review.

Flows

- f1 Review comments.
- f2 Password: Personal password for the PC-members, used both for security, authentication, and user-tailoring of the screens.
- f3 PC-member password.
- f4 PC-member password.
- f5 Papers to review.
- f6 Existing password.
- f7 Personalia: Sent to make the review form customized.
- f8 Paperlist.
- f9 Wrong password: Error message.
- f10 Old review: A previous review in case edit has been chosen.
- f11 Review setup: To create the web-form dynamically.
- f12 Paper to review: Indicate the paper to review.
- f13 Review to delete.
- f14 Paperno, pcmemberno.
- f15 Delete review.
- f16 Reviewform.
- f17 Update paperdistribution: In case of a delete of the review.
- f18 Enable entering of review: Indicate the temporal relationship between the two processes.
- f19 Review by web.

- f20 Reviewing not successful: Error message.
- f21 Update review.
- f22 Confirmation on reviewing.
- f23 Update paper distribution.
- f24 Review by post.
- f25 Review by fax.
- f26 Review by mail.
- f27 Review.
- f28 Error: Errors in the registration of the review.
- f29 Update distribution.
- f30 New review.
- f31 Confirmation on review.
- f32 Update review.
- f33 Old review.
- f34 New review.
- f35 Successful update.
- f36 Error in update.
- f37 Review to delete.
- f38 Delete review.
- f39 Update distribution.
- f40 Successful delete.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R151			R108	
f2		R108		P4.1		
f3	P4.2			P4.1		
f4	P4.3			P4.1		
f5	S10			P4.1		
f6	S9			P4.1		
f7	P4.1			P4.2		
f8	P4.1				R108	
f9	P4.1				R108	
f10	S13			P4.2		
f11	S12			P4.2		
f12		R108		P4.2		
f13		R108		P4.2		
f14	P4.3			P4.2		
f15	P4.2			S13		
f16	P4.2				R108	
f17	P4.2			S10		
f18	P4.2			P4.3		
f19		R108		P4.3		
f20	P4.3				R108	
f21	P4.3			S13		
f22	P4.3				R108	

continued on next page

continued from previous page						
	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f23	P4.3			S10		
f24		R108		A102	R112	A22
f25		R108		A102	R112	A22
f26		R108		A102	R112	A22
f27	A102	R112	A22	P4.4		
f28	P4.4.			A102	R112	A22
f29	P4.4			S10		
f30	P4.4			S13		
f31	P4.4			A102	R112	A22
f32	A102	R112	A22	P4.5		
f33	S13			P4.5		
f34	P4.5			S13		
f35	P4.5			A102	R112	A22
f36	P4.5			A102	R112	A22
f37	A102	R112	A22	P4.6		
f38	P4.6			S13		
f39	P4.6			S10		
f40	P4.6			A102	R112	A22

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A255			P4.1			Potential	
A255			P4.2			Potential	
A255			P4.3			Potential	
A257			P4.4			Potential	
A257			P4.6			Potential	

Version 2

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A255			P4.1				
A255			P4.2				
A255			P4.3				
A257			P4.4				
A257			P4.5			Potential	
A257			P4.6				

The diagram for the issuing of reminders on reviews are given in Figure A.33. It indicates that a general notification is sent 2 weeks before the deadline, and a more specific reminder when the deadline is passed.

Actors

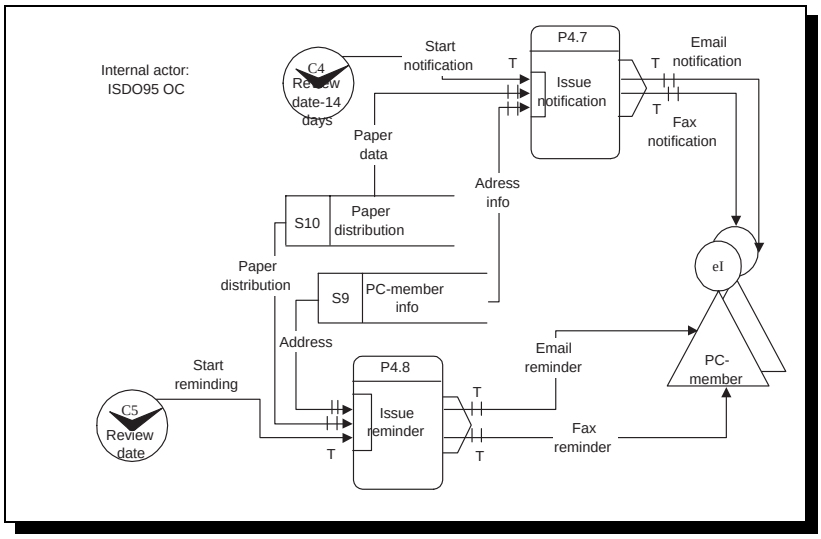

Figure A.33: PPM describing new reminding of reviews

- A22 ISDO95: External organizational.
- A102 JK: Internal individual.
- A230 C4: Internal applicative.

Roles

- R108 PC-member: External individual.
- R112 Program coordinator: Internal individual.

Stores

- S9 PC-member info.
- S10 Paper distribution: Indicate which PC-members that are responsible for reviewing which papers.

Timers

- C4 Reviewdate - 14 days.
- C5 Reviewdate.

Processes

- P4.7 Issue notification: Issues a gentle notification that it is two weeks to the review deadline. Do not issue the notification to those already having sent in all their reviews.
- P4.8 Issue reminder: Issues a more concrete reminder to all who have not submitted all their reviews.

Flows

- f1 Start reminding: Signal.
- f2 Paper distribution.
- f3 Address: of PC-member.
- f4 Email reminder: Sent to those having email.
- f5 Fax reminder: Sent to those not having email.
- f6 Start notification: Signal.
- f7 Paper data.
- f8 Address info.
- f9 Email notification.
- f10 Fax notification: To those not having email.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	C5			P4.8		
f2	S10			P4.8		
f3	S9			P4.8		
f4	P4.8				R108	
f5	P4.8				R108	
f6	C4			P4.7		
f7	S10			P4.7		
f8	S9			P4.7		
f9	P4.7				R108	
f10	P4.7				R108	

Support relationships
Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A102	R112	A22	P4.7				
A102	R112	A22	P4.8				

Version 2

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A230			P4.7			Potential	
A230			P4.8			Potential	

Process P5: Selection of papers :

This is divided in two diagrams, one for the actual selection, and one for sending the notification to the contributers. The diagram indicating the selection process is given in Figure A.34 and is similar to the one presented in the model of the perceived current situation.

Actors

- A22 ISDO95: External organizational.
- A102 JK: Internal individual.

Stores

- S5 Paperinfo.
- S6 Manual paper: Store of physical papers.
- S14 PC-reports: Review forms including the confidential part and summary statistics.
- S16 Sessions.

Timers

- C6 PC-meeting.

Processes

- P5.1 Select papers: As part of the PC-meeting.
- P5.2 Create evaluation: Creation of different reports based on the reviews.

Flows

- f1 Session info: Information about decided sessions.

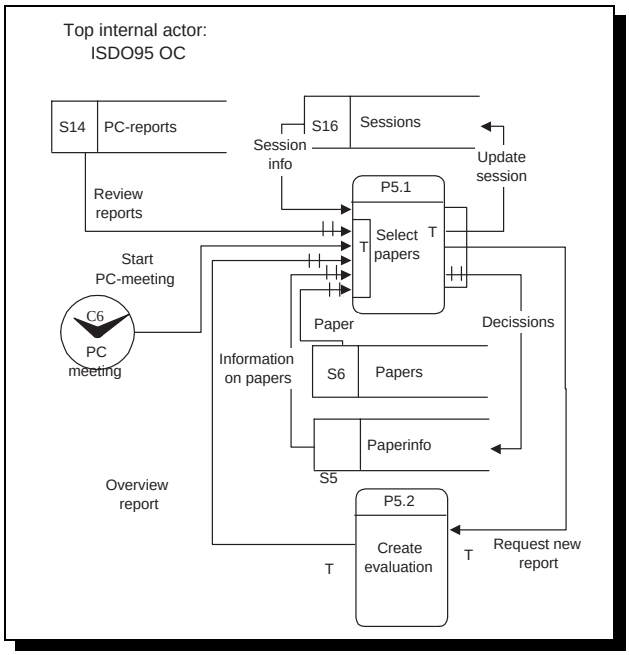

Figure A.34: PPM describing new selection of papers

- f2 Review report.
- f3 Start PC-meeting: Signal.
- f4 Overview report.
- f5 Information on papers.
- f6 Paper.
- f7 Update session.
- f8 Request new report.
- f9 Decisions: Update the status of the paper as they are accepted or rejected.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	S16			P5.1		
f2	S14			P5.1		
f3	C6			P5.1		
f4	P5.2			P5.1		
f5	S5			P5.1		
f6	S6			P5.1		
f7	P5.1			S16		
f8	P5.1			P5,2		
f9	P5.1			S5		

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A22			P5.1				
A102	R112		P5.2				

Version 2 is equal to version 1.

The diagram for issuing notification of results of the selection is given in Figure A.35. For those having an accepted paper, one also issue CRC guidelines and a copyright-form.

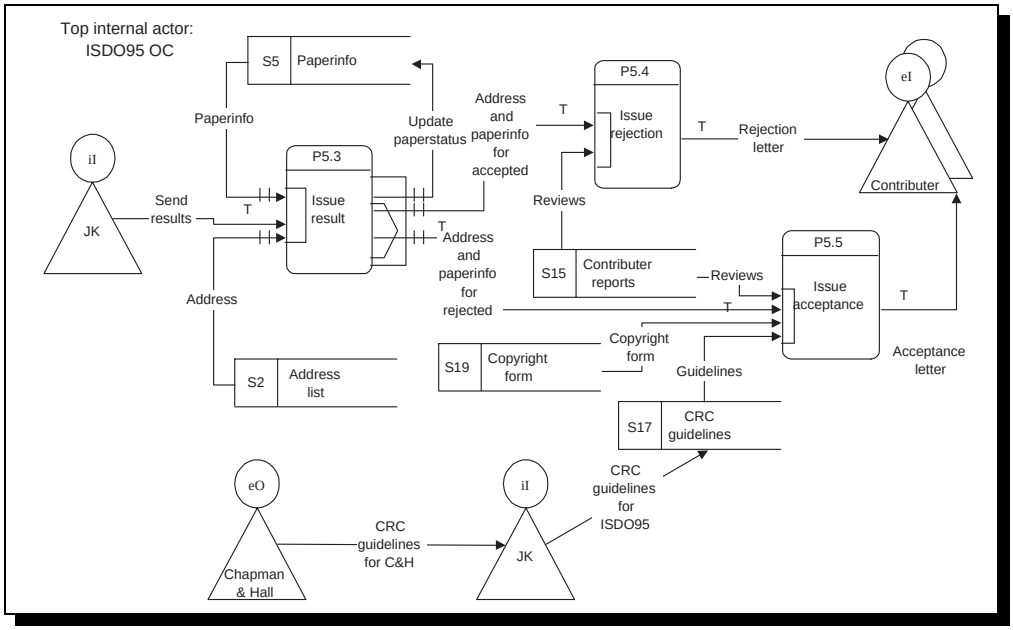

Figure A.35: PPM describing new notification of results

Actors

- A22 ISDO95: External organizational.
- A40 Chapman & Hall: External organizational.
- A102 JK: Internal individual.
- A230 C4: Internal applicative.

Roles

- R112 Program coordinator: Internal individual.
- R149 Contributer: External individual. Someone potentially contributing a paper to the conference.

Stores

- S2 Address list: List of addresses for those that are to receive the call for papers.
- S5 Paperinfo.
- S15 Contributer reports: Review forms only including the public part of this, and not the confidential part.
- S17 CRC guidelines: Guidelines for the layout of the CRC.
- S19 Copyright form: A form to fill in to transfer copyright of the material to the publisher.

Processes

- P5.3 Issue result: Issue the result of the paper selection to the contributers.
- P5.4 Issue rejection.
- P5.5 Issue acceptance.

Flows

- f1 Send results: Signal.
- f2 Paperinfo.

- f3 Address: of the contributer.
- f4 Update paperstatus.
- f5 Address and paperinfo for accepted.
- f6 Address and paperinfo for rejected.
- f7 Reviews.
- f8 Rejection letter: Include the review reports.
- f9 Reviews.
- f10 Copyright form.
- f11 Guidelines.
- f12 Acceptance letter: Includes the review reports, CRC guidelines, and copyright form.
- f13 CRC guidelines for C&H: The IFIP working conference proceedings guidelines used at Chapman & Hall.
- f14 CRC guidelines for ISDO95.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	A102	R112	A22	P5.3		
f2	S5			P5.3		
f3	S2			P5.3		
f4	P5.3			S5		
f5	P5.3			P5.4		
f6	P5.3			P5.5		
f7	S15			P5.4		
f8	P5.4				R149	
f9	S15			P5.5		
f10	S19			P5.5		
f11	S17			P5.5		
f12	P5.5				R149	
f13	A40	R3		A102	R112	A22
f14	A102	R112	A22	S17		

Support relationships
Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A102	R112	A22	P5.3				
A102	R112	A22	P5.4				
A102	R112	A22	P5.5				

Version 2

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A230			P5.3			Potential	
A230			P5.4			Potential	
A230			P5.5			Potential	

Process P5.2: Creation of review-reports :

The diagram is given in Figure A.36. It indicates that report can be made for the reviews of one paper in forms suiting both the PC and the contributers. In addition it is possible to present overview-reports.

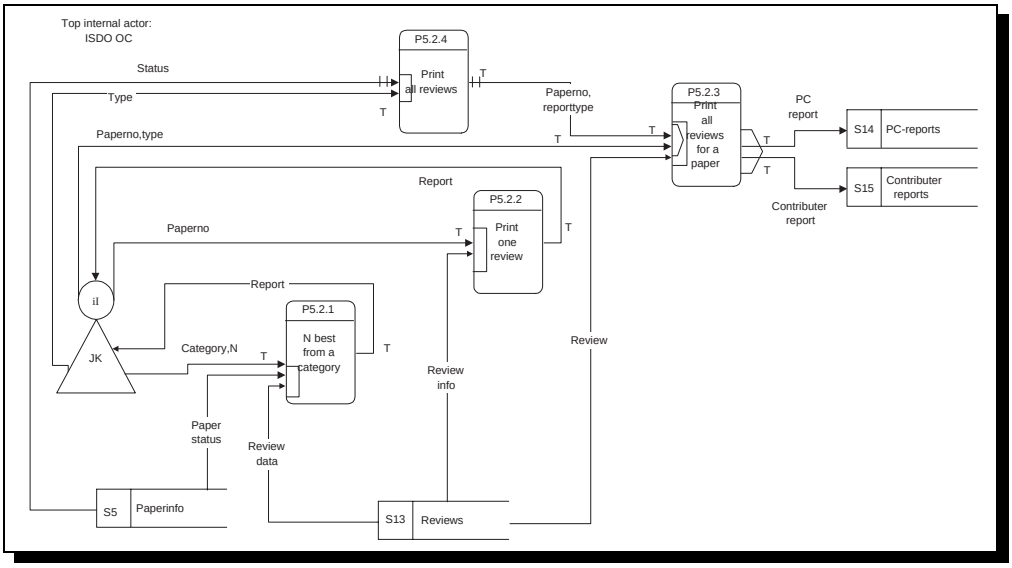

Figure A.36: PPM describing create review-reports

Actors

- A22 ISDO95: External organizational.
- A102 JK: Internal individual.
- A256 C4shell: Internal applicative. The Unix-shell part of C4.
- A257 C4motif: Internal applicative. The Motif part of C4.

Roles

- R112 Program coordinator: Internal individual.

Stores

- S5 Paperinfo.
- S13 Reviews: Evaluations of papers.
- S14 PC-reports: Review forms including the confidential part and summary statistics.
- S15 Contributer reports: Review forms only including the public part of this, and not the confidential part.

Processes

- P5.2.1 N best from a category: Produce a report with the N best papers according to a given category (overall, originality etc).
- P5.2.2 Print one review.
- P5.2.3 Print all reviews: Print all reviews from all papers.
- P5.2.4 Print all reviews for a paper: Type of review is indicated, i.e. PC or contributer report.

Flows

- f1 Category,N.
- f2 Paperstatus: Indicate if the paper is accepted, rejected or undecided.
- f3 Review data.

- f4 Report.
- f5 Paperno.
- f6 Review info.
- f7 Report.
- f8 Paperno,type: Paper number and type of report.
- f9 Review.
- f10 Paperno, reporttype.
- f11 PC report.
- f12 Contributer report.
- f13 Type.
- f14 Status: Paperstatus indicating if all reviews are received or not.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	A102	R112	A22	P5.2.1		
f2	S5			P5.2.1		
f3	S13			P5.2.1		
f4	P.5.2.1			A102	R112	A22
f5	A102	R112	A22	P5.2.2		
f6	S13			P5.2.2		
f7	P5.2.2			A102	R112	A22
f8	A102	R112	A22	P5.2.3		
f9	S13			P5.2.3		
f10	P5.2.4			P5.2.3		
f11	P5.2.3			S14		
f12	P5.2.3			S15		
f13	A102	R112	A22	P5.2.4		
f14	S5			P5.2.4		

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A257			P5.2.1			Potential	

Version 2

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A257			P5.2.1				
A256			P5.2.2			Potential	
A256			P5.2.3			Potential	
A256			P5.2.4			Potential	

Process P6: Reception of CRC: Also this is divided into two diagrams, one for the reception and checking of the CRC, and one for sending notifications:

The diagram for the reception and checking of the CRC is given in Figure A.37. It indicates the possibility of also issuing the CRC by ftp. Papers are checked to see if they are updated according to comments and that they have the correct layout. If OK, the CRC is registered, if not a message is sent to the contributer to fix the problems and resend the paper.

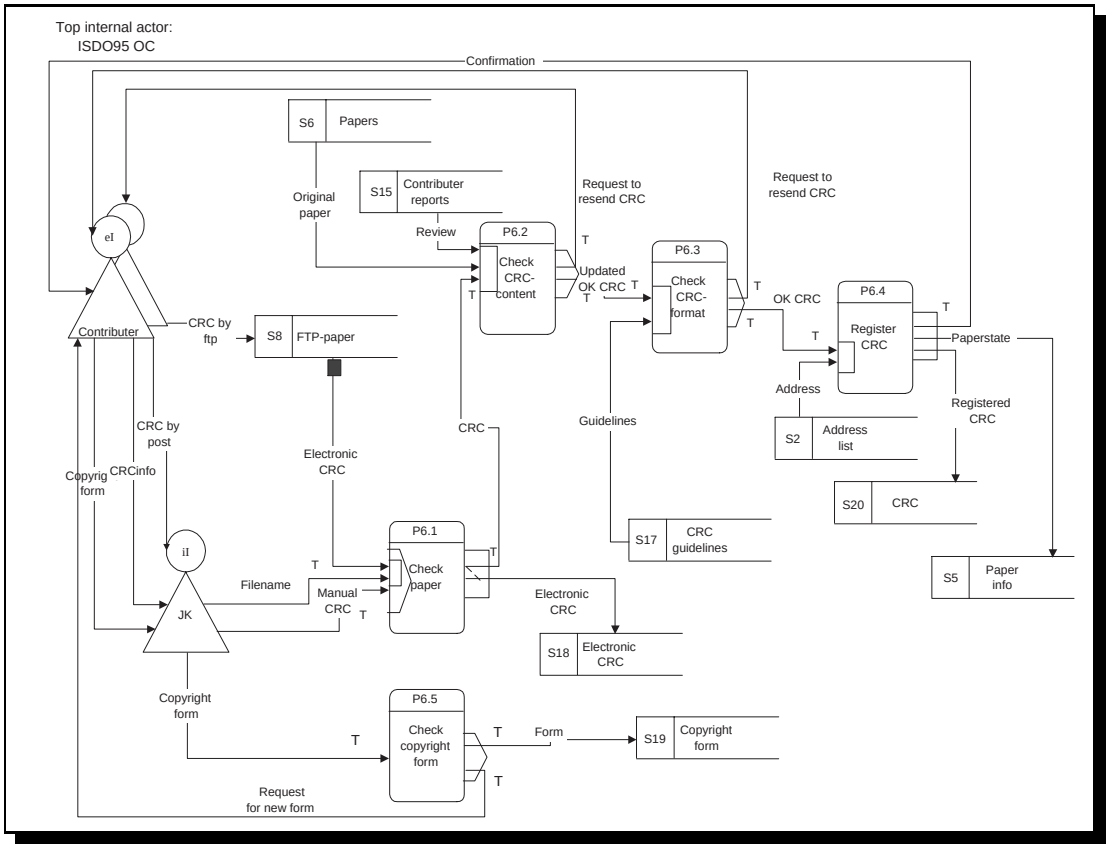

Figure A.37: PPM describing new reception of CRC

Actors

- A22 ISDO95: External organizational.
- A102 JK: Internal individual.

Roles

- R112 Program coordinator: Internal individual.
- R149 Contributer: External individual. Someone potentially contributing a paper to the conference.

Stores

- S2 Address list: List of addresses for those that are to submitt CRC.
- S5 PC-member info.
- S6 Manual paper: Store of physical papers.
- S8 FTP-paper: Ftp-directory.
- S15 Contributer reports: Review forms only including the public part of this, and not the confidential part.
- S17 CRC guidelines: Guidelines for the layout of the CRC.

- S18 Electronic CRC: Directory containing electronic CRCs.
- S19 Copyright form: A form to fill in to transfer copyright of the material to the publisher.
- S20 CRC: Paper CRCs.

Processes

- P6.1 Check paper: If the paper is electronic, it is checked that it can be printed out.
- P6.2 Check CRC-content: Check that the paper is changed according to the review comments.
- P6.3 Check CRC-format: Check that the paper follows the layout guidelines.
- P6.4 Register CRC.
- P6.5 Check copyright form.

Flows

- f1 CRC by ftp.
- f2 Filename: Filename of submitted CRC.
- f3 Electronic CRC.
- f4 Manual CRC.
- f5 CRC.
- f6 Electronic CRC.
- f7 Original paper: The paper that was submitted originally.
- f8 Review.
- f9 Request to resend CRC: In case there is something wrong with the content.
- f10 Updated OK CRC.
- f11 Guidelines: for layout.
- f12 Request to resend CRC: because of non-conformant layout.
- f13 OK CRC.
- f14 Address: Address of the contributer.
- f15 Confirmation: of the reception of an OK CRC.
- f16 Paperstate.
- f17 Registered CRC.
- f18 Copyright form.
- f19 Form.
- f20 Request for new form: if the form is not correctly filled out.
- f21 CRC by post.
- f22 CRCinfo: Notification typically by email that the CRC has been submitted to the ftp-directory.
- f23 Copyright form.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R149		S8		
f2	A102	R112	A22	P6.1		
f3	S8			P6.1		
f4	A102	R112	A22	P6.1		
f5	P6.1			P6.2		
f6	P6.1			S18		
f7	S6			P6.2		
f8	S15			P6.2		

continued on next page

continued from previous page						
	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f9	P6.2				R149	
f10	P6.2			P6.3		
f11	S17			P6.3		
f12	P6.3				R149	
f13	P6.3			P6.4		
f14	S2			P6.4		
f15	P6.4				R149	
f16	P6.4			S5		
f17	P6.4			S20		
f18	A102	R112	A22	P6.5		
f19	P6.5			S19		
f20	P6.5				R149	
f21		R149		A102	R112	A22
f22		R149		A102	R112	A22
f23		R149		A102	R112	A22

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A102	R112	A22	P6.1				
A102	R112	A22	P6.2				
A102	R112	A22	P6.3				
A257	R112		P6.4				
A102	R112	A22	P6.5				

Version 2 equal to version 1.

The diagram for reminder of CRC is given in Figure A.38.

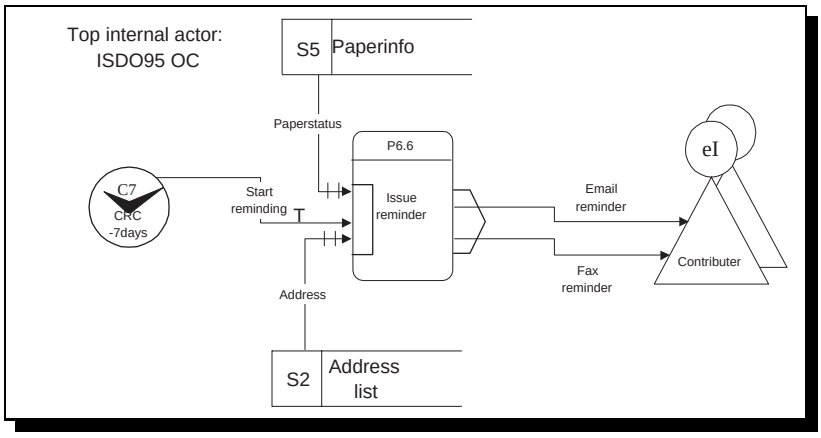

Figure A.38: PPM describing new reminding on CRC

Actors

- A22 ISDO95: External organizational.
- A102 JK: Internal individual.
- A230 C4: Internal applicative.

Roles

- R112 Program coordinator: Internal individual.
- R149 Contributer: External individual. Someone potentially contributing a paper to the conference.

Stores

- S2 Address list: List of addresses for those that are to receive the call for papers.
- S5 Paperinfo.

Timers

- C7 CRC - 7 days.

Processes

- P6.6 Issue reminder.

Flows

- f1 Start reminding: Signal.
- f2 Paperstatus: To find which papers have not been yet received.
- f3 Address: of contributer.
- f4 Email reminder.
- f5 Fax reminder: For those not having email.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	C1			P6.6		
f2	S5			P6.6		
f3	S2			P6.6		
f4	P6.6				R149	
f5	P6.6				R149	

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A102	R112	A22	P6.6				

Version 2

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A230			P6.6			Potential	

Process P7: Creation of manuscript :

The diagram is given in Figure A.39. It indicates that sources of data that are used in the preparation of the manuscript. At the deadline for issuing the manuscript, it is sent to Chapman & Hall, which return the finished proceedings before the conference.

Actors

- A22 ISDO95: External organizational.

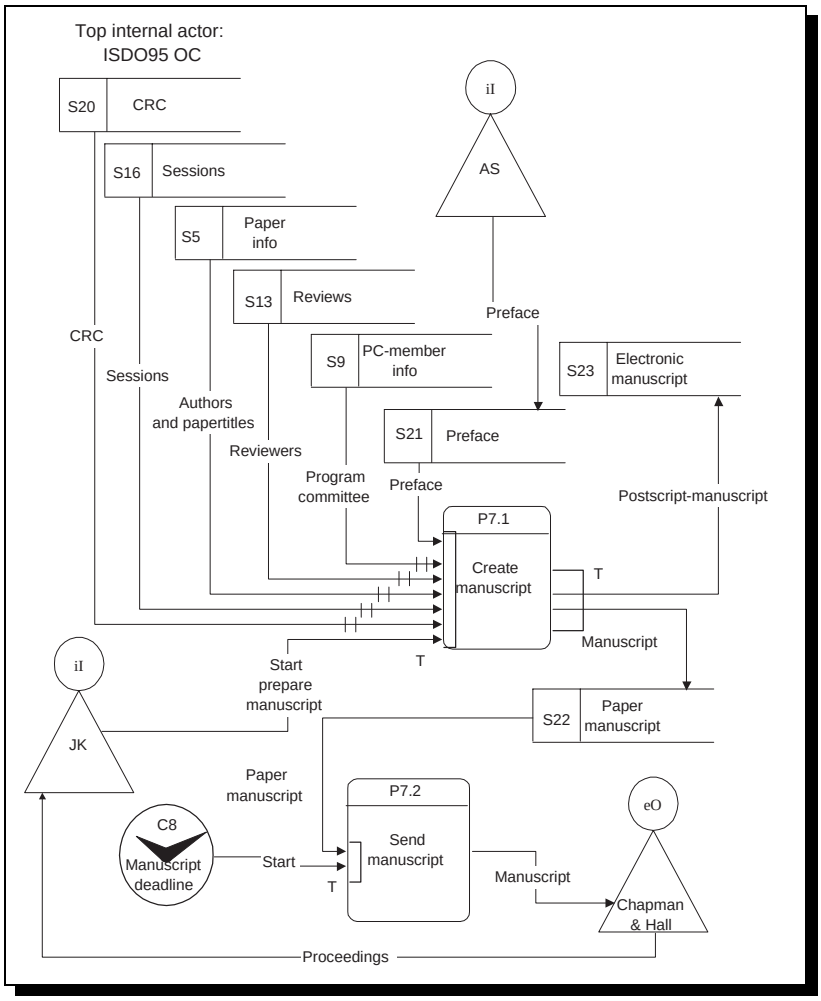

Figure A.39: PPM describing new preparation of manuscript

- A40 Chapman & Hall: External organizational.
- A101 AS: Internal individual.
- A102 JK: Internal individual.

Roles

- R3 Publisher: External organizational. Publisher of proceedings of a conference.
- R112 Program coordinator: External individual.
- R145 Proceedings editor: Internal individual.
- R146 Program chair: External individual.

Stores

- S5 Paperinfo.
- S9 PC-member info.
- S13 Reviews: Evaluations of papers.
- S16 Sessions.
- S20 CRC.
- S21 Preface: Preface of proceedings.
- S22 Paper manuscript.

Timers

- C8 Manuscript deadline.

Processes

- P7.1 Create manuscript.
- P7.2 Send manuscript: Send the created manuscript to the publisher.

Flows

- f1 Start prepare manuscript: Signal.
- f2 CRC.
- f3 Sessions: Decided sessions of the conference to order the CRCs in the same order.
- f4 Authors and papertitles: To the table of contents.
- f5 Reviewers: To overview of reviewers of the conference.
- f6 Program committee. List over the members of the program committees.
- f7 Preface.
- f8 Postscript manuscript.
- f9 Manuscript: Paper manuscript.
- f10 Start: Signal.
- f11 Paper manuscript.
- f12 Manuscript.
- f13 Proceedings.
- f14 Preface.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1	A102	R112	A22	P7.1		
f2	S20			P7.1		
f3	S16			P7.1		
f4	S5			P7.1		
f5	S13			P7.1		
f6	S9			P7.1		
f7	S21			P7.1		
f8	P7.1			S23		
f9	P7.1			S22		
f10	C8			P7.2		
f11	S22			P7.2		
f12	P7.2			A40	R3	
f13	A40	R3		A102	R145	A22
f14	A101	R146	A22	S21		

Support relationships

Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A102	R112	A22	P7.1				
A102	R112	A22	P7.2				

Version 2 equal to version 1.

Process P8: Address register :

For this process, we have also indicate the detailed descriptions of processes. The diagram is given in Figure A.40, and support both the direct addition of address-information via WWW, and the local maintenance of the address register. An the address of a person already exist it is updated with the new data.

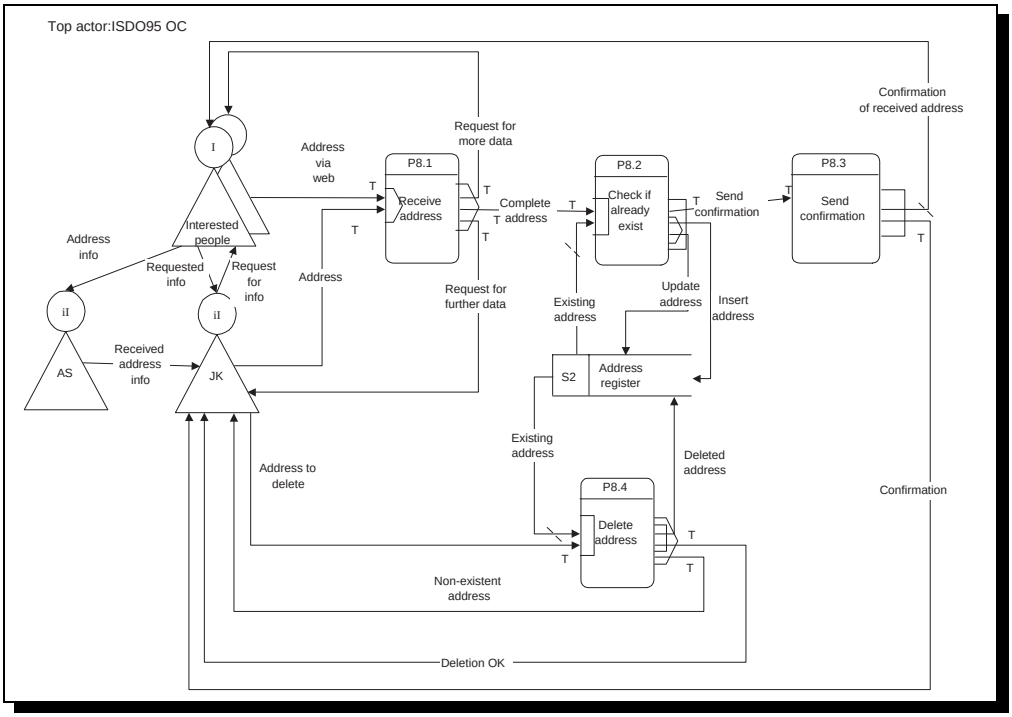

Figure A.40: PPM describing registration and update of addresses

Actors

- A101 AS : Individual actor internal to IDT being program chair of the ISDO95 conference.
- A102 JK : Individual internal actor to IDT being program coordinator of the ISDO95 conference.
- A230 C4: Internal applicative.

Roles

- R153 Interested people: Individual giving notice of their interest of the conference. Can be both internal and external. Can also be other people that one want to have updated information on.

Stores

- S2 Address-register: The part of the data-base including personalia. The view to the data-model is given in Figure A.41

Processes

- P8.1 Receive address: Receives an address, and decides if the address is sufficient, i.e. contains sufficient information. A PLD for this process is given in Figure A.42.
  DRL-rules for the processing of this process is given below:
  R590

  ```
  When address_via_web(Firstname,Initials,Lastname,Title,Phone,Fax,Email,
  Organization,Address,Country)
  ```

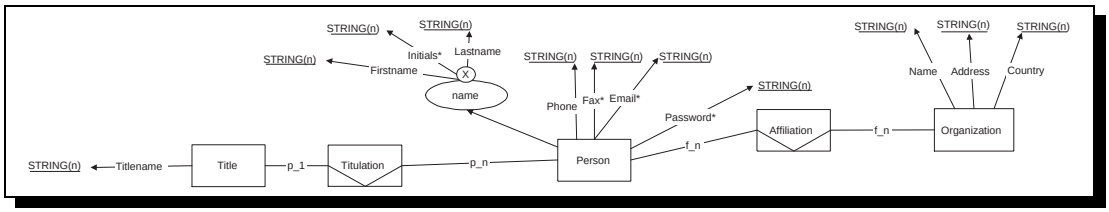

Figure A.41: Scenario to ONER-diagram relevant to the address-register

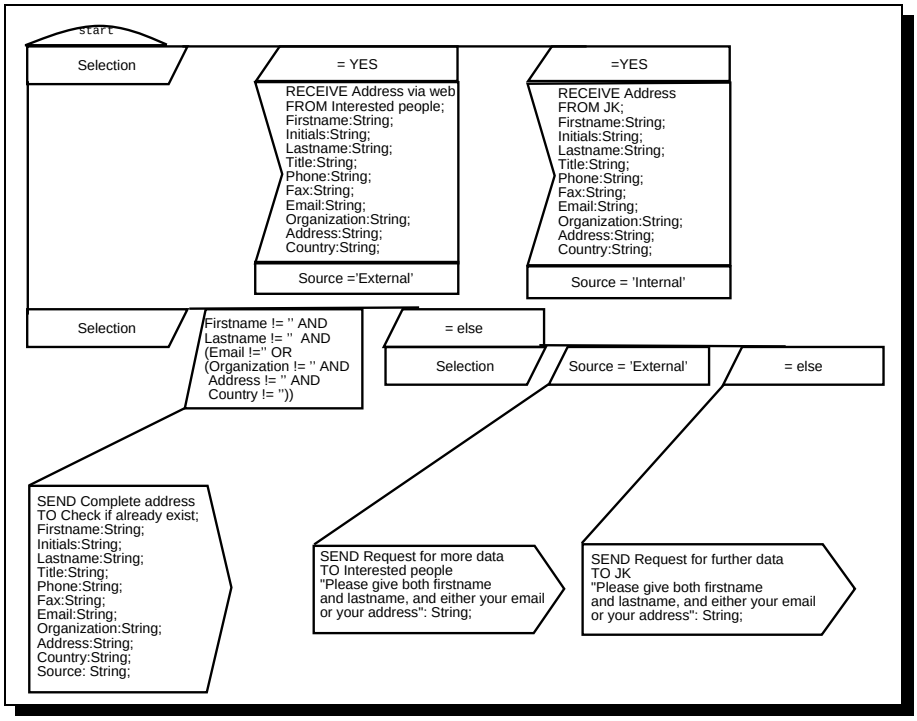

Figure A.42: PLD for Process 8.1: Receive address

```
if Firstname <> ‘’’’ and Lastname <>’’’’
and (Email <> ‘’’’ or (Organization <> ‘’’’ and Address <> ‘’’’ and
Country <> ‘’’’))
then it is obligatory for the system
complete_address(Firstname,Initials,Lastname,
Title,Phone,Fax,Email,Organization,Address.Country,’’External’’).
else it is obligatory for the system
request_for_more_data(‘‘Please give both firstname and lastname,
and either your email or your address’’).
```

R591

```
When address(Firstname,Initials,Lastname,Title,Phone,Fax,Email,
Organization,Address,Country)
if Firstname <> ‘’’’ and Lastname <>’’’’
and (Email <> ‘’’’ or (Organization <> ‘’’’ and Address <> ‘’’’ and
Country <> ‘’’’))
then it is obligatory for the system
complete_address(Firstname,Initials,Lastname,
```

```
Title,Phone,Fax,Email,Organization,Address.Country,''Internal'').
else  it is obligatory for the system
request_for_further_data(``Please give both firstname and lastname,
and either your email or your address'').
```

- P8.2 Check existence of address: If an address already exist in the address register this is updated, else the new address information is inserted. A PLD for this process is given in Figure A.43.

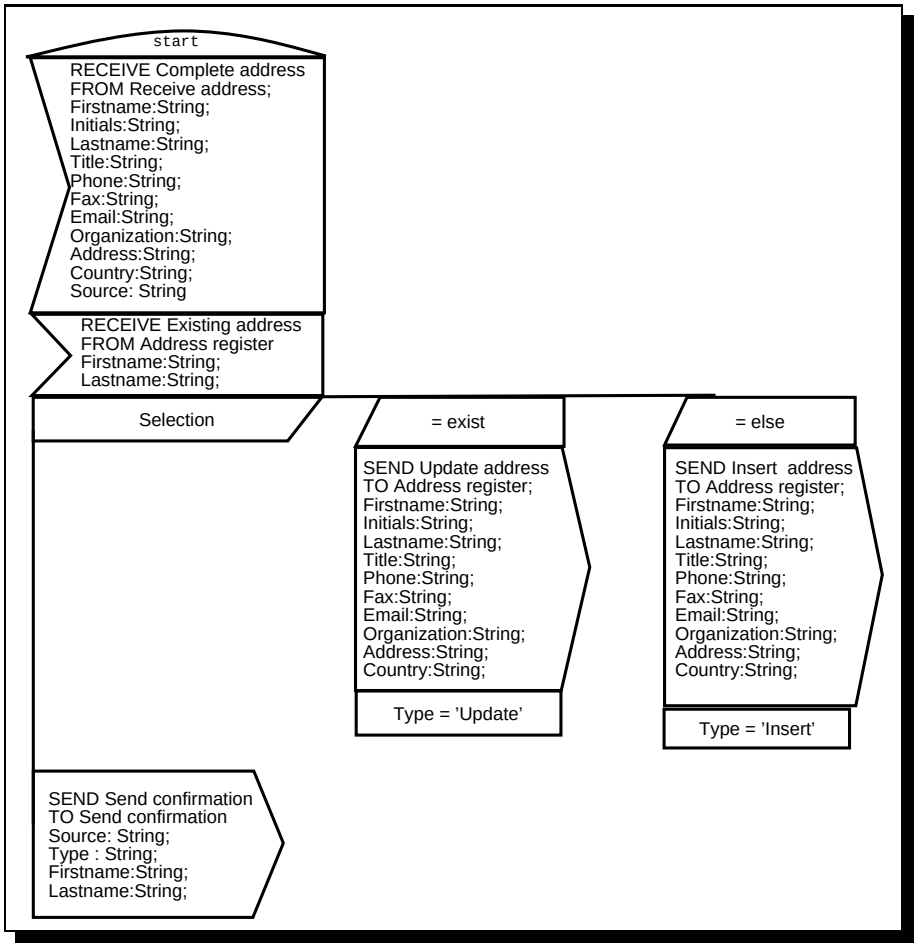

Figure A.43: PLD for Process 8.2: Check if already exist

DRL-rules for the processing of this process is given below:
R592

```
When complete_address((Firstname,Initials,Lastname,
Title,Phone,Fax,Email,Organization,Address.Country,Source)
if person [has name [has firstname(Firstname),
                     has lastname(Lastname)]]
then it is obligatory for the system
person[titulation(title[has titlename(Title)]),
       has name [has firstname(Firstname),
                 has initials(Initials),
                 has lastname(Lastname)].
```

```
      has phone(Phone),
      has fax(Fax),
 has email(Email),
 affiliation(organization[has name(Organization),
                          has address(Address),
                          has country(Country)])
     ]
and send_confirmation(Source,''Update'', Firstname, Lastname).
else it is obligatory for the system
person[titulation(title[has titlename(Title)]),
      has name [has firstname(Firstname),
                has initials(Initials),
                has lastname(Lastname)].
      has phone(Phone),
      has fax(Fax),
 has email(Email),
 affiliation(organization[has name(Organization),
                          has address(Address),
                          has country(Country)])
     ]
and send_confirmation(Source,''Insert'', Firstname, Lastname).
```

- P8.3 Send confirmation: After a successful update has taken place, a confirmation of the registration is issued. A PLD for this process is given in Figure A.44.

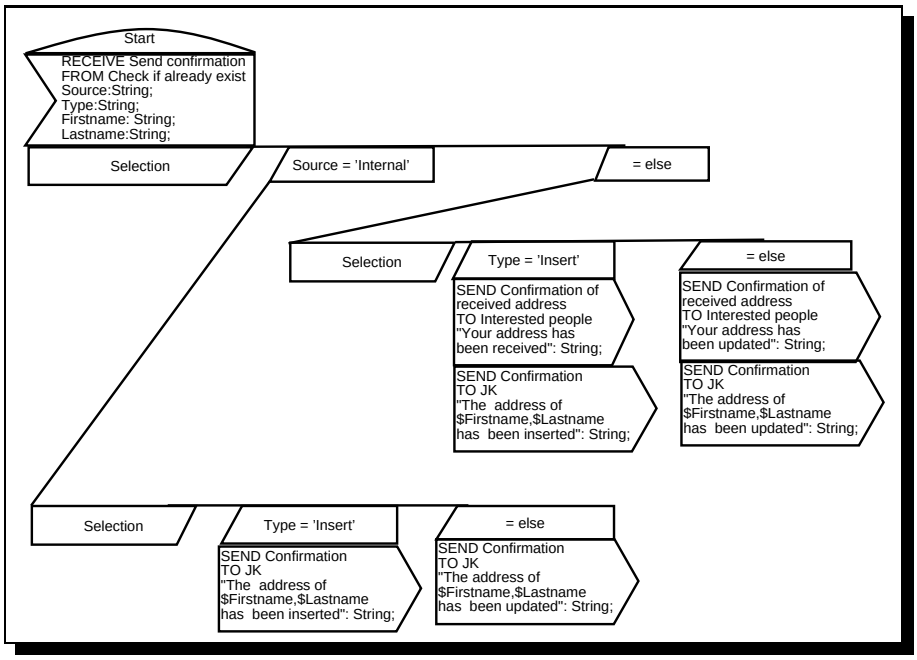

Figure A.44: PLD for Process 8.3: Send confirmation

DRL-rules for the processing of this process is given below:

R593

```
when send_confirmation(Source,Type,Firstname,Lastname)
if Type =''Insert''
then it is obligatory for the system
confirmation(‘‘The address of ''Firstname''''Lastname'' has been inserted'')
```

R594

```
when send_confirmation(Source,Type,Firstname,Lastname)
if Type =''Update''
then it is obligatory for the system
confirmation(‘‘The address of ''Firstname''''Lastname'' has been
updated'')
```

R595

```
when send_confirmation(Source,Type,Firstname,Lastname)
if Source=''External'' and Type =''Insert''
then it is obligatory for the system
confirmation_of_received_address(‘‘Your address have been inserted'');
```

R596

```
when send_confirmation(Source,Type,Firstname,Lastname)
if Source=''External'' and Type =''Update''
then it is obligatory for the system
confirmation_of_received_address(‘‘Your address have been
updated'');
```

- P8.4. Delete address: Delete an address from the address register.
  A PLD for this process is given in Figure A.45.

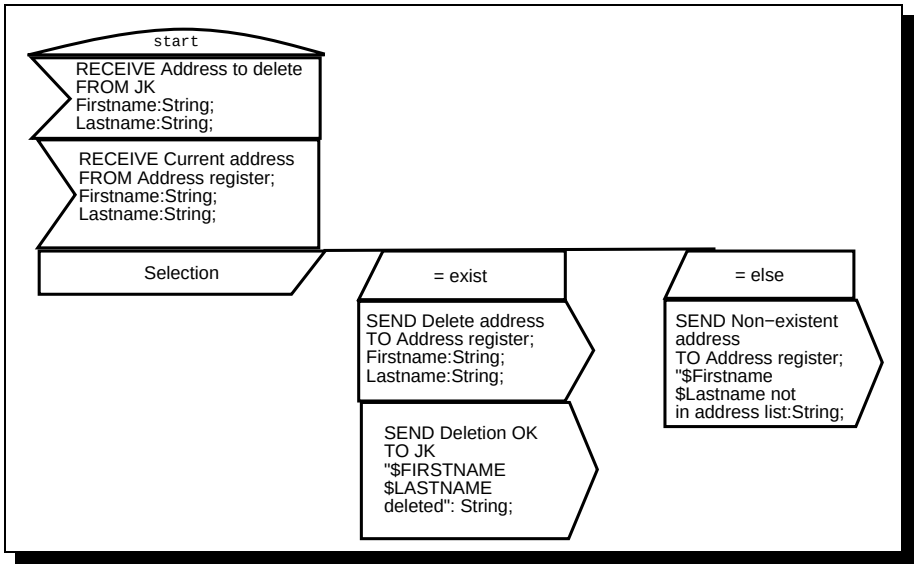

Figure A.45: PLD for Process 8.4: Delete address

DRL-rules for the processing of this process is given below:
R597

```
When address_to_delete(Firstname,Lastname)
if person [has name [has firstname(Firstname),
                    has lastname(Lastname)]]
then it is permitted for Program coordinator
not (person [has name [has firstname(Firstname),
                    has lastname(Lastname)]]).
and deletion_ok(Firstname Lastname'' deleted'')
```

R598

```
When address_to_delete(Firstname,Lastname)
if person [has name [has firstname(Firstname),
                    has lastname(Lastname)]]
then it is obligatory for the system
non-existent_address(Firstname Lastname'' not in address list'')
```

Flows

- f1 Address via web: Address sent in directly via WWW from interested people.
- f2 Address: Address information
- f3 Request for more data: Sent if an address is received by web, but do not contain sufficient data.
- f4 Complete address: Address that has been checked to contain sufficient information.
- f5 Request for further data: Sent if an address is tried entered in the local program, but do not contain sufficient data.
- f6 Existing address: Name.
- f7 Send confirmation: Start the process that confirms the entering of the address and notifies JK if the address is inserted by WWW.
- f8 Insert address: Address of person that was not registered earlier.
- f9 Update address: Address of person that was registered earlier.
- f10 Confirmation of received address: Confirmation to web-user.
- f11 Confirmation: Confirmation of received address either via web or internally.
- f12 Address to delete: Name to delete from address register.
- f13 Current address: Name registered in the database.
- f14 Deleted address: Delete the address of a person.
- f15 Deletion OK: Notification that the deletion has been successful.
- f16 Non-existent address: Message in the case one tries to delete an address that do not exist.
- f17 Address-information: Address information received by AS.
- f18 Revived address info: Address information sent from AS to JK.
- f19 Requested info: Address information sent from interested people to JK.
- f20 Request for info: Request for address information.

	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f1		R153		P8.1		
f2	A102			P8.1		
f3	P8.1				R153	
f4	P8.1			P8.2		
f5	P8.1			A102		

continued on next page

continued from previous page						
	From			*To*		
Flow	Node	Role	Behalf	Node	Role	Behalf
f6	S2			P8.2		
f7	P8.2			P8.3		
f8	P8.2			S2		
f9	P8.2			S2		
f10	P8.3				R153	
f11	P8.3			A102		
f12	A102			P8.4		
f13	S2			P8.4		
f14	P8.4			S2		
f15	P8.4			A102		
f16	P8.4			A102		
f17		R153		A101		
f18	A101			A102		
f19		R153		A102		
f20	A102				R153	

Support relationships
Version 1

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A230			P8.1			Potential	
A230			P8.2			Potential	
A230			P8.3			Potential	
A230			P8.4			Potential	

Version 2 equal to version 1 with potential changed to actual.
Figure A.46 indicates an actor model for the application system. Top internal actor is IDT.
Actors

- A222 WWW: Internal supportive.
- A224 Ingres database: Internal supportive.
- A226 Motif: Internal supportive.
- A227 make: Internal supportive.
- A229 gcc: Internal supportive.
- A230 C4: Internal applicative.
- A255 C4 WWW: Internal applicative.
- A256 C4 Shell: Internal applicative.
- A257 C4 Motif: Internal applicative.

Relationship	From	Role	Behalf	To	Comment
Part-of	A255			A230	
Part-of	A256			A230	
Part-of	A257			A230	

Support relationships

Supporter			*Supported*				
Actor	Role	Behalf	Node	Role	Behalf	Status	Comment
A222			A255				
A224			A255				
A224			A256				
A224			A257				
A226			A257				
A227			A255				
A227			A257				
A229			A257				

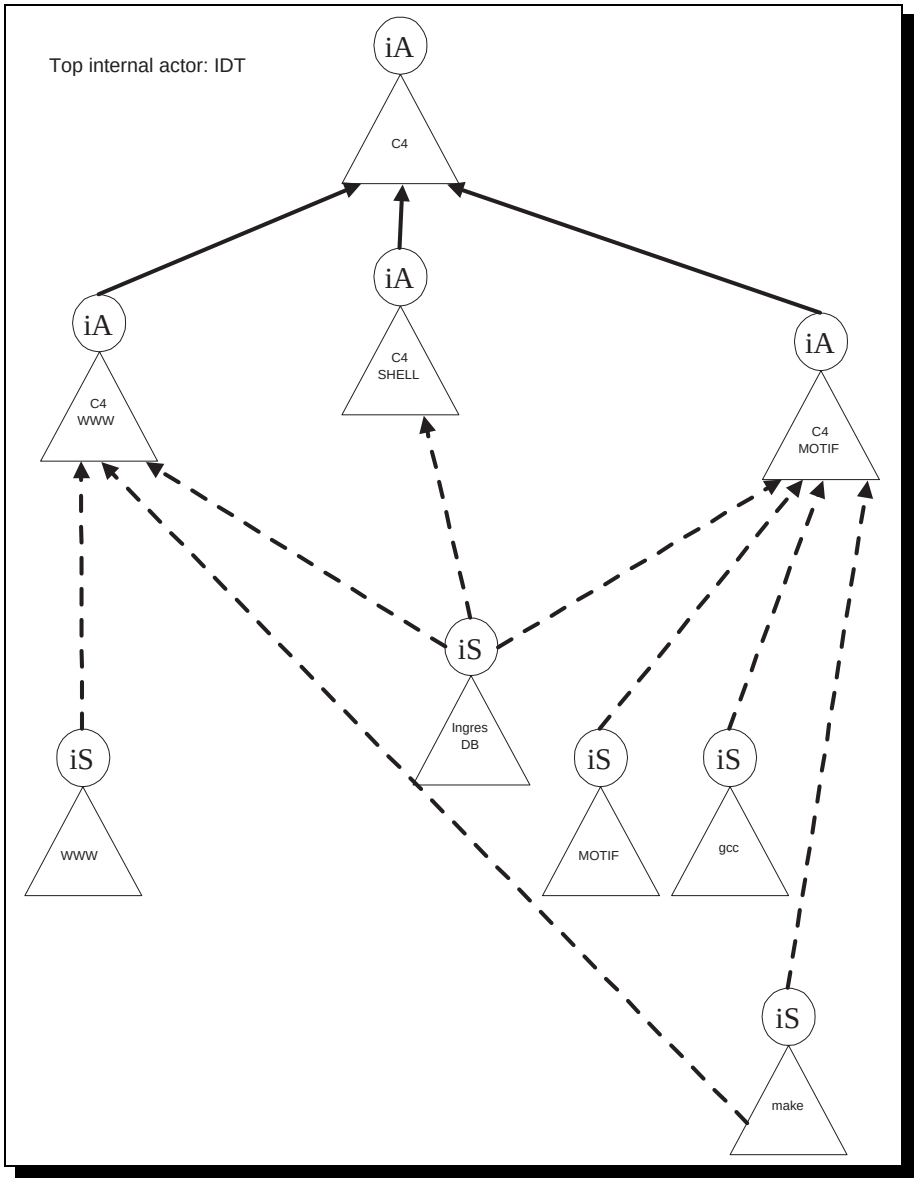

Figure A.46: Actor model describing the C4 application system

Rules in the conference case

We give here an overview of rules from conference case-study, the starting initials are the primary source of the rule.

The rules are divided in two parts: General rules for a professional conference, and specific rules applying to the future IS and CIS. This do not mean that all rules for the first area appeared in the modeling of the perceived current situation. Still we have found it beneficial to divide the rules in this way.

Within these groups, the rules are again divided according to the following areas:

- General rules.
- Personalia.
- CFP.
- Papers.
- Reviews.
- PC-meetings.
- CRC.
- Participation.
- Practical organization.
- Proceedings.

In addition is there a set of rules being project management oriented at the end. Only some of the links to the data and process models are given. The links are to the model of the perceived future IS. Areas which were not regarded further in the projects are marked with 'PRUNED'.

General rules for a professional conference

General :

1 Rule (*AS* **)**
A research group has limited resources

R1 *obligates* **R6** *AS*

2 Rule (*AS* **)**
It is obligatory for a research community to spread scientific results.

R2 *obligates* **R3** *AS*
R2 *recommends* **R351** *AS*
R2 *recommends* **R510** *JK* **A** It is more and more usual that the research community get their information using e.g. WWW.

3 Rule (*AS* **)**
It is recommended for research groups to arrange successful conferences.

R3 *obligates* **R4** *AS*
R3 *obligates* **R17** *JK*
R3 *recommends* **R401** *OIL* **A** Such an organization have experience in the practical arrangement of such conferences.
R3 *recommends* **R453** *JK*
R3 *recommends* **R501** *AS*
R3 *recommends* **R504** *AS* **A** Many professional conferences are arranged. If we can

come up with a general system that can be used in other conferences arranged by the research community, this will be a valuable contribution from us to the community.

R3 *obligated by* **R2**

4 Rule (*AS* **)**
It is obligatory for the organizing committee to create interest for the conference.

R4 *obligates* **R8** *AS*
R4 *recommends* **R9** *OIL*
R4 *related to* **R5** *JK*

R4 *obligated by* **R3**

5 Rule (*JK* **)**
It is recommended for the organizing committee to create interest for the conference.

6 Rule (*AS* **)**
It is obligatory for the organizing committee to use an adequate amount of resources on the conference.

R6 *obligates* **R219** *JK*
R6 *recommends* **R351** *OIL*
R6 *discourages* **R401** *AS* **A** It is costly to use the services of an external organization.
R6 *recommends* **R452** *AS*
R6 *recommends* **R508** *JK*
R6 *recommends* **R654** *JK* **A** It is more costly to support many formats.
R6 *discourages* **R512** *JK*

1 Ruleset
R3 *and* **R6**

RS1 *recommends* **R7**

7 Rule (*AS* **)**
It is obligatory for the organizing committee to create a budget proposal.

8 Rule (*AS* **)**
It is obligatory to have an interesting and high-quality technical program on a conference.

R8 *obligates* **R10** *AS*
R8 *recommends* **R13** *AS*
R8 *recommends* **R14** *AS*
R8 *recommends* **R15** *AS*
R8 *recommends* **R16** *AS*
R8 *recommends* **R651** *AS* **A** The members of the program committee can motivate their research groups directly to issue papers. With a large program committee, one can still assure a fair review of the papers.

R8 *obligated by* **R4**

9 Rule (*OIL* **)**
It is recommended for a conference oirganizer to have an interesting social program of a conference.

R9 *obligated by* **R4**

PRUNED

10 Rule (*JK* **)**
It is obligatory to have paper-sessions on a conference with high quality papers.

R10 *recommends* **R11** *AS*
R10 *obligates* **R12** *AS*
R10 *obligates* **R101** *JK*
R10 *obligates* **R151** *JK*
R10 *obligates* **R202** *AS*
R10 *recommends* **R254** *AS*
R10 *recommends* **R302** *JK*
R10 *recommends* **R303** *JK*
R10 *obligates* **R355** *JK*

R10 *obligated by* **R8**

11 Rule (*AS* **)**
It is recommended that the papers in a session are related.

R11 *obligates* **R213** *JK*

R11 *recommended by* **R10**

12 Rule (*AS* **)**
It is recommended for the conference organizers to get many papers submitted to the conference.

R12 *recommends* **R102** *AS*
R12 *recommends* **R103** *AS*
R12 *obligates* **R657** *JK* **A** This is the usual way of receiving papers.

R12 *obligated by* **R10**

13 Rule (*AS* **)**
It is recommended to have invited speakers on a conference.

R13 *recommended by* **R8**

PRUNED

14 Rule (*AS* **)**
It is recommended to have panel discussions on a conference.

R14 *recommended by* **R8**

PRUNED

15 Rule (*AS* **)**
It is permitted to have poster sessions on a conference.

R15 *recommended by* **R8**

PRUNED

16 Rule (*AS* **)**
It is recommended to have tutorials on a conference.

R16 *recommended by* **R8**

PRUNED

17 Rule (*JK* **)**
It is recommended for the organizing committee to have good working relationships with all stakeholders of the conference.

R17 *obligates* **RS2** *JK*
R17 *recommends* **R23** *JK* **A** It is irritating to receive the same message many times.
R17 *obligates* **R655** *JK* **A** There are still many that do not have email.
R17 *recommends* **R512** *JK* **A** If the database goes down, it should still be possible to do the main tasks.

R17 *obligated by* **R3**

18 Rule (*JK* **)**
It is obligatory for an organizing committee to have good working relationships with the contributers to the conference.

R18 *recommends* **R154** *JK*
R18 *obligates* **R156** *JK*
R18 *obligates* **R218** *JK*
R18 *obligates* **R657** *JK* **A** This is the ordinary way of submitting papers that most potential contributers are used to.

19 Rule (*AS* **)**
It is recommended for an organizing committee to have good working relationships with the publisher of the proceedings.

R19 *recommended by* **R456**

20 Rule (*AS* **)**
It is obligatory for an organizing committee to have good working relationships with the program committee members.

R20 *recommends* **R220** *AS*
R20 *obligates* **R702** *JK* **A** Not all PC-members have good electronic communication.
R20 *obligates* **R706** *JK* **A** Not all PC-members have email.

21 Rule (*OIL* **)**
It is obligatory for an organizing committee to have good working relationships with the conference organization.

R21 *discourages* **R905** *OIL*

22 Rule (*OIL* **)**
It is recommended for a conference organizer to have good working relationships with the participants of the conference.

R22 *recommends* **R952** *JK* **A** The participants of a conference expect to be given a paper proceedings.

2 Ruleset
R18 *and* **R19** *and* **R20***and* **R21** *and* **R22**

RS2 *obligated by* **R17**

23 Rule (*JK* **)**
It is recommended to not send the same message to the same person more than once.

R23 *obligates* **R24** *JK*

R23 *recommended by* **R17**

24 Rule (*JK* **)**
It is recommended to have a list over messages sent to different persons.

R24 *obligates* **R554** *JK*
R24 *obligates* **R555** *JK* **A** Often when using electronic communication, the message is returned undeliverred.

R24 *obligated by* **R24**

Personalia :

51 Rule (*AS* **)**
It is recommended to have support for updating and maintenance of address registers.

R51 *recommends* **R53** *AS*
R51 *recommends* **R55** *JK*
R51 *recommends* **R56** AS

R51 *obligated by* **R102**

52 Rule (*AS* **)**
It is recommended to have sufficient info in the database on persons that possibly will be interested in a conference.

R52 *obligates* **R556** *JK*
R52 *recommends* **R557** *AS*
R52 *recommends* **R558** *AS*

R52 *recommended by* **R102**

53 Rule (*AS* **)**
It is permitted to buy addresses from available sources.

R53 *related-to* **R54**

R53 *recommended by* **R51**

54 Rule (*AS* **)**
It is recommended for the OC to buy addresses from available sources.

R54 *related-to* **R53**

PRUNED

55 Rule (*JK* **)**
It is recommended to update address-registers by using e-mail servers.

R55 *recommended by* **R51**

PRUNED

56 Rule (*AS* **)**
It is recommended to send email to persons to ask for updated information.

R56 *recommended by* **R51**

PRUNED

CFP :

101 Rule (*AS* **)**
It is obligatory for the program chair to create a CFP containing necessary information for prosepective contributers.

R101 *obligates* **R105** *JK*
R101 *obligates* **R106** *AS*
R101 *obligates* **R107** *JK*
R101 *recommends* **R108** *JK*
R101 *recommends* **R109** *JK*
R101 *obligates* **R110** *AS*
R101 *recommends* **R111** *JK*

R101 *obligated by* **R10**
R101 *necessitated by* **R102**
R101 *necessitated by* **R103**

102 Rule (*AS* **)**
It is obligatory for the OC to spread the CFP widely.

R102 *obligates* **R51** *AS*
R102 *recommends* **R52** *AS*
R102 *necessitates* **R101** *AS*
R102 *recommends* **R551** *JK*
R102 *recommends* **R552** *AS*
R102 *recommends* **R553** *JK*

R102 *recommended by* **R12**

Related to P1.2 and P1.3.

103 Rule (*JK* **,** *AS* **)**
It is obligatory for a PC-member to distribute the call for paper widely before the CFP-deadline.

R103 *necessitates* **R104** *JK*

R103 *recommended by* **R12**

104 Rule (*AS* **)**
It is recommended for the OC to give the CFP to the PC-members.

R104 *necessitates* **R101** *JK*

Related to P1.3.

R105 *obligated by* **R101**

105 Rule (*JK* **)**
It is obligatory that a CFP contains the important deadlines of the conference.

Related to P1.1.

106 Rule (*AS* **)**
It is obligatory to inform about the area of the conference in the CFP.

R106 *obligated by* **R101**

Related to P1.1.

107 Rule (*JK* **)**
It is obligatory to indicate in the CFP the expected format and number of copies of the contribution.

R107 *obligated by* **R101**
R107 *obligated by* **R155**

Related to P1.1.

108 Rule (*JK* **)**
It is recommended to indicate in the CFP how the review process will take place.

R108 *recommend by* **R101**

Related to P1.1.

109 Rule (*AS* **)**
It is obligatory to list the members of the program-committee in the CFP.

R109 *recommended by* **R101**

Related to P1.1.

110 Rule (*AS* **)**
It is obligatory to list the organizing committee in a CFP

R110 *obligated by* **R101**

Related to P1.1.

111 Rule (*JK* **)**
It is recommended to have a list of keywords that an author can use to classify his paper.

R111 *obligates* **R215** *JK*

R111 *recommended by* **R101**

Related to P1.1 (PRUNED).

Papers :

151 Rule (*JK* **,** *AS* **)**
It is obligatory for the conference organizers to be able to receive papers.

R151 *obligates* **R152** *JK*

R151 *obligated by* **R10**

152 Rule (*JK* **)**
It is obligatory for the program coordinator to keep track of papers.

R152 *obligated by* **R151**

Related to P2.2.

153 Rule (*AS* **)**
It is forbidden for members of the organizing committee to submit papers to the conference.

R153 *obligated by* **R202**

Related to E5 and E8.

154 Rule (*JK* **)**
It is recommended for the program coordinator to confirm the reception of papers.

R154 *recommends* **R18**

Related to P2.3.

155 Rule (*JK* **)**
It is obligatory for someone issuing papers on paper to issue it in the requested number of copies.

R155 *obligates* **R107** *JK*

R155 *recommended by* **R202**

156 Rule (*JK* **)**
It is forbidden for the organizing committee to make any papers available for any others than the program committee.

R156 *obligate* **R713** *JK*

R156 *obligated by* **R18**

157 Rule (*JK* **)**
If before CFP-deadline, it is permitted for an IS-professional to issue a paper to the conference.

158 Rule (*AS* **)**
It is permitted for an author to withdraw a submitted paper.

R158 *recommends* **R223**

159 Rule (*JK* **)**
It is forbidden for an IS-professional to submitt the same paper to several conferences and/or journals at the same time.

Reviews :

201 Rule (*JK* **)**
It is obligatory for the program coordinator to keep track of reviews of papers.

R201 *obligated by* **R202**

Related to P4.

202 Rule (*JK* **,** *AS* **)**
It is obligatory to be able to have papers properly quality-checked.

R202 *obligates* **R153** *AS* **A** To ensure independent reviewing.
R202 *recommends* **R155** *JK*
R202 *obligates* **R 201** *JK*
R202 *obligates* **R203** *AS*
R202 *recommends* **R204** *JK*
R202 *obligates* **R205** *AS* **A** To ensure independent reviewing.
R202 *recommends* **R206** *AS*
R202 *recommends* **R207** *AS* **A** To make the workload bearable for PC-members.
R202 *recommends* **R208** *AS* **A** The other person might be more knowledgeable.
R202 *recommends* **R209** *AS* **A** It should appear on the PC-meeting who has been performing the review.
R202 *recommends* **R210** *JK*
R202 *recommends* **R212** *AS* **A** Different research cultures on different continents and countries can be anticipated.
R202 *recommends* **R217** *JK* **A** To ensure independant reviewing.
R202 *recommends* **R211** *AS*
R202 *recommends* **RS251** *AS*
R202 *recommends* **R253** *AS*
R202 *recommends* **R757** *AS*
R202 *discourages* **R651** *JK*
R202 *obligates* **R701** *JK*
R202 *recommends* **R758** *JK*
R202 *recommends* **R222** *JK*

R202 *obligated by* **R10**

203 Rule (*JK* **,** *AS* **)**
It is obligatory to distribute papers to reviewers.

R203 *obligated by* **R202**

Related to P3.3.

204 Rule (*JK* **,** *AS* **)**
If being given papers for reviews, it is obligatory for a PC-member to review the papers and send the reviews to the organizing committee before the review deadline.

R204 *recommended by* **R202**

205 Rule (*AS* **)**
It is forbidden for PC-members to review papers they are authors of.

R205 *obligated by* **R202**

Related to P3.2 and E6 Rel6 E12 Rel10 E18 Rel9 E8.

206 Rule (*AS* **)**
It is obligatory for a conference organizer to have each paper reviewed by at least three reviewers.

R206 *recommended by* **R202**

Related to P3.1 and P3.2.

207 Rule (*AS* **)**
It is discouraged for a conference organizer to give each PC-member more than 8 papers to review.

R207 *recommended by* **R202**

Related to P3.1.

208 Rule (*AS* **)**
It is permitted for a PC-member to have someone else review the paper.

R208 *recommended by* **R202**

Related to P3.

209 Rule (*AS* **)**
If someone else than the PC-member review the paper it is obligatory for the PC-member to report their names to the OC.

R209 *recommended by* **R202**

Related to E6 Rel6 E12 Rel7 E7.

210 Rule (*AS* **)**
It is recommended for the organizing committee to give papers to PC-members which are knowledgeable in the area the paper is about.

R210 *obligates* **RS201** *JK*

R210 *recommended by* **R202**

Related to P3.2.

211 Rule (*AS* **)**
It is recommended that the reviewer can indicate his knowledge of the area on the review-form.

R211 *recommended by* **R202**

Related to E17

212 Rule (*AS* **)**
It is recommended to give papers to PC-members in different areas of the world

R212 *recommended by* **R202**

Related to P3.2 and E6 Rel2 E2.

213 Rule (*JK* **)**
It is recommended to have papers classified using keywords.

R213 *recommends* **R215** *JK*

Related to E18 Rel21 E19.

214 Rule (*JK* **)**
It is recommended to have expertise among PC-members classified using keywords.

R214 *recommends* **R216** *JK*

Related to E6 Rel23 E19.

215 Rule (*JK* **)**
It is recommended to create a list of keywords which paper-writers can use when classifying their paper.

R215 *obligated by* **R111**
R215 *recommended by* **R213**

216 Rule (*JK* **)**
It is recommended to ask PC-members within which keyword-areas they like to review papers.

R216 *recommended by* **R214**

201 Ruleset
R215 *and* **R216**

RS201 *obligated by* **R210**

217 Rule (*JK* **)**
It is forbidden to give a paper for review to a PC-member if he is in the same research-group as one of the authors.

R217 *recommended by* **R202**

Related to E2 Rel2 E6 Rel6 E12 Rel10 E18 Rel9 E8 Rel2 E2.

218 Rule (*JK* **)**
If a person has submitted a paper to the conference it is obligatory for the conference organizers to send the verdict of the review, together with the actual review to the contact-person of the paper within the notification-limit.

R218 *obligates* **R756** *JK*

R218 *obligated by* **R18**

219 Rule (*JK* **)**
It is obligatory to only send the review-result to the contact-person of the paper.

R219 *related to* **R218**

R219 *recommended by* **R6**

Related to P5.3.

220 Rule (*JK* **)**
It is discouraged to let reviewers see the detailed reviews of other reviewers.

R220 *recommends* **R713**

R220 *recommended by* **R20**

221 Rule (*JK* **)**
It is obligatory for the organizing committee to make sure of the source of the review.

R221 *obligates* **R713**

222 Rule (*JK* **)**
It is recommended for a PC-member to notice the OC immediately if they do not think they are able to give a paper a proper review.

223 Rule (*JK* **)**
When a paper is witdrawn, it is recommended for the conference organizer to notice the reviewers of the paper.

R223 *recommended by* **R158**

PC-meetings :

251 Rule (*AS* **)**
It is recommended for a PC-member to attend the program committee meeting.

R251 *related to* **R252**

R251 *recommended by* **R202**

252 Rule (*JK* **)**
It is obligatory for a PC-member to attend the program committee meeting.

R252 *related to* **R251**

253 Rule (*JK* **)**
It is obligatory for the program coordinator to make a report of the reviews.

R253 *recommends* **R753** *JK*
R253 *obligates* **R754** *JK*
R253 *obligates* **R755** *JK*

R253 *recommended by* **R202**

251 Ruleset
R253 *and* **R218**

RS251 *obligates* **R756** *JK*

254 Rule (*AS* **)**
It is obligatory to let each paper have at least 30 minutes for presentation and discussion.

R254 *recommended by* **R10**

CRC :

301 Rule (*JK* **)**
If they want their papers published then it is obligatory for those with accepted papers to issue the CRC before the CRC deadline.

R301 *obligated by* **R451**

302 Rule (*JK* **)**
If they have received conditional accept then it is obligatory for authors to change their paper according to the review-comments if they have received conditional accept.

R302 *related to* **R304** *JK*

R302 *recommended by* **R10**

303 Rule (*JK* **)**
If their paper is accepted then it is recommended for authors to change their paper according to the review-comments.

R303 *recommended by* **R10**

304 Rule (*JK* **)**
It is obligatory for the program chair to check that when people have got conditional accept, they have changed their papers according to the comments.

R304 *related to* **R305** *JK*

305 Rule (*JK* **)**
It is obligatory for the program chair to supply the author with description of the CRC-format.

R305 *obligated by* **R455**

Related to R5.5.

301 Ruleset
R455 *and* **R18**

RS301 *recommends* **R306** *JK*

306 Rule (*JK* **)**
It is recommended to supply the author with electronic templates to make the CRC according to the description.

R306 *obligates* **R307** *JK*
R306 *recommends* **R308** *JK*
R306 *recommends* **R309** *JK*

R306 *recommended by* **RS301**

307 Rule (*JK* **)**
It is obligatory to support Latex-format.

R307 *obligated by* **R306**

308 Rule (*JK* **)**
It is recommended to support Word-format.

R308 *recommended by* **R306**

309 Rule (*JK* **)**
It is recommended to support Framemaker-format.

R309 *recommended by* **R306**

PRUNED.

310 Rule (*JK* **)**
It is obligatory for an accepted author to supply a written statement about copyright together with the CRC to have the paper published in the proceedings.

R310 *obligated by* **R456**

Related to P6.5.

Participation :

351 Rule (*OIL* **)**
It is obligatory to get many participants to a conference.

R351 *recommends* **R353** *AS*
R351 *obligates* **R352** *OIL*
R351 *recommends* **R354** *AS*
R351 *recommends* **R851** *JK*
R351 *obligates* **R852** *JK* **A** WWW is being used by more and more people to get information about scientific and other events.

R351 *recommended by* **R2**
R351 *recommended by* **R6**

352 Rule (*OIL* **)**
It is obligatory for the organizing committee to create a call for participation.

R352 *obligated by* **R351**
R352 *necessitated by* **R353**

353 Rule (*JK* **,** *AS* **)**
It is obligatory for a PC-member to distribute the call for participation widely before the conference.

R353 *necessitates* **R352** *JK*

R353 *recommended by* **R351**

354 Rule (*AS* **)**
It is recommended for a PC-member of a conference to attend the conference.

R354 *recommended by* **R351**

355 Rule (*JK* **)**
It is obligatory that one of the authors of an accepted paper attend the conference to present the paper.

R355 *obligated by* **R10**

Practical organization :

401 Rule (*OIL* **)**
It is recommended for an organizing committee to apply the services of a local conference organization.

R401 *obligates* **R902** *OIL* **A** SEVU is the local conference organizer.

R401 *recommended by* **R3**
R401 *discouraged by* **R6**

401 Ruleset
R7 *and* **R401**

RS401 *obligates* **R402** *OIL*

402 Rule (*OIL* **)**
It is obligatory for an OC to make a contract with the external organization.

R402 *obligated by* **RS401**

Proceedings :

451 Rule (*AS* **)**
It is obligatory for the program committee to publish the result of a professional conference in the form of a proceedings.

R451 *obligates* **R301** *AS*
R451 *obligates* **R454** *JK*

R451 *necessitated by* **R452**
R451 *necessitated by* **R453**

452 Rule (*AS* **)**
It is obligatory to get paid for the proceedings.

R452 *necessitates* **R451** *AS*

R452 *recommended by* **R6**

451 Ruleset
R2 *and* **R452**

RS451 *recommends* **R456** *AS* **A** To have a distribution channel for the results.

453 Rule (*JK* **)**
It is obligatory that the proceedings are available at the conference.

R453 *necessitates* **R451** *JK*

R453 *recommended by* **R3**

454 Rule (*JK* **)**
It is discouraged for the OC to publish the proceedings of the conference without all CRCs.

R454 *obligated by* **R451**

455 Rule (*JK* **)**
It is recommended to have a uniform format of the finished proceedings.

R455 *obligates* **R305** *JK*

456 Rule (*JK* **)**
It is recommended to have a professional publisher to publish the proceedings.

R456 *recommends* **R19** *JK*
R456 *obligates* **R310** *JK*
R456 *forbids* **R951** *JK* **A** At least as long as the publisher do not publish electronically.
R456 *obligates* **R952**

R456 *recommended by* **RS451**

Rules applying to the conference system

General :

501 Rule (*AS* **)**
It is recommended for the IS-group to arrange the ISDO95 conference.

R501 *recommended by* **R3**
R501 *necessitated by* **R502**

502 Rule (*JK* **)**
It is obligatory that the conference system can be used at the ISDO95 conference.

R502 *necessitates* **R501** *JK*
R502 *recommends* **R503** *JK* **A** One should support shorter turnaround-time from CFP-deadline to the conference.
R502 *obligates* **R505** *JK* **A** I will be the main user of the system, and I will be using a Unix-workstation.
R502 *obligates* **R663** *JK*
R502 *recommends* **R751** *AS*
R502 *recommends* **R759** *AS*
R502 *recommends* **R901** *OIL*

501 Ruleset
R502 *and* **R304**

RS501 *obligates* **R801** *JK*

502 Ruleset
R502 *and* **R454**

RS502 *obligates* **R802** *JK*

503 Rule (*JK* **)**
It is obligatory for the conference system to support electronic availability of information.

R503 *recommends* **R510** *JK*

R503 *recommended by* **R502**

504 Rule (*AS* **)**
It is recommended for the conference system to support the organization of other conferences than ISDO95.

R504 *recommends* **R506** *AS* **A** Many people in the research community use PCs.
R504 *recommends* **R507** *AS* **A** Several people in the research community use Macintosh.
R504 *recommends* **R716** *JK*

R504 *recommended by* **R3**

505 Rule (*JK* **)**
It is obligatory that the conference system run on UNIX.

R505 *obligated by* **R502**

506 Rule (*AS* **)**
It is recommended that the conference system run on PC using MS-Windows.

R506 *recommended by* **R504**
R506 *obligated by* **R752**

PRUNED

507 Rule (*AS* **)**
It is recommended that the conference system run on Macintosh.

R507 *recommended by* **R504**

PRUNED

503 Ruleset
R505 *and* **R506** *and* **R507**

504 Ruleset
R505 *or* **R506** *or* **R507**

RS504 *obligated by* **R1004**

508 Rule (*JK* **)**
It is recommended to transfer as much as possible of the correspondence from paper to electronic form.

R508 *obligates* **R509** *JK AS*
R508 *recommends* **R510** *JK OIL*
R508 *recommends* **R511** *JK* **A** ftp is the fastest and most reliable way of transferring large files electronically.
R508 *recommends* **R703** *JK*
R508 *recommends* **R704** *JK*
R508 *recommends* **R904** *OIL*
R508 *recommends* **R905** *OIL*
R508 *recommends* **R951** *AS*

R508 *recommended by* **R6**

505 Ruleset
R17 *and* **R508**

RS505 *obligates* **R656** *JK*

509 Rule (*JK* **)**
It is obligatory to use email.

R509 *obligates* **R556** *JK*

R509 *obligated by* **R508**

510 Rule (*JK* **)**
It is obligatory to use WWW.

R510 *recommended by* **R2**
R510 *recommended by* **R503**
R510 *recommended by* **R508**
R510 *necessitated by* **R601**

506 Ruleset
R510 *and* **R102**

RS506 *recommends* **R601** *JK*

511 Rule (*JK* **)**
It is recommended to use ftp.

R511 *recommended by* **R508**
R511 *recommended by* **R652**

512 Rule (*JK* **)**
It is recommended for the conference system to support several ways of performing important tasks.

R512 *discouraged by* **R6** **R512** *recommended by* **R17**

Personalia :

551 Rule (*JK* **)**
It is permitted for persons to register information about themselves by WWW.

R551 *recommended by* **R102**

Related to P8.1.

551 Ruleset (*JK* **)**
R17 *and* **R102**

RS551 *recommends* **R590** *GR*
RS551 *recommends* **R595** *GR*
RS551 *recommends* **R596** *GR*

552 Rule (*AS* **)**
It is obligatory to support importing addresses from other sources in the system.

R552 *recommended by* **R102**

PRUNED.

553 Rule (*JK* **)**
It is obligatory to be able to update data imported to the system.

R553 *recommended by* **R102**

554 Rule (*JK* **)**
It is obligatory for the conference system to keep track of messages send from the system.

R554 *obligated by* **R24**

PRUNED

555 Rule (*JK* **)**
It is obligatory to be able to manually change the overview of who has received a given message.

R555 *obligated by* **R24**

PRUNED.

556 Rule (*JK* **)**
It is obligatory to be able to store both postal and email addresses and fax numbers on persons.

R556 *obligates* **R591**

R556 *obligated by* **R52**

Related to P8.1. and E1.

552 Ruleset (*JK* **)**
R556 *and* **R551**

RS552 *obligates* **R590** *GR*

557 Rule (*AS* **)**
It is recommended to be able to store a correctness assessment on addresses.

R557 *recommended by* **R52**

PRUNED.

558 Rule (*AS* **)**
It is recommended to save when an address was last updated.

R558 *recommended by* **R52**

PRUNED.

For rules **R590** to **R598**, see under P8.

CFP :

601 Rule (*JK* **)**
It is recommended to make the CFP available on WWW.

R601 *necessitate* **R510** *JK*

R601 *recommended by* **RS506**

Related to P1.2.

602 Rule (JK)
It is obligatory to indicate in the CFP the acceptable format of an electronic paper.

R602 *obligated by* **R660**

Papers :

651 Rule (*AS* **)**
It is permitted for members of the ISDO95 PC to submit papers to ISDO95.

R651 *recommended by* **R8**
R651 *recommended by* **R202**

652 Rule (*JK* **)**
It is recommended to be able to receive electronic papers.

R652 *recommends* **R511** *JK* **A** The safest and fastest way of transferring large files are by using ftp.
R652 *obligates* **R653** *JK*
R652 *recommends* **R659** *JK*

R652 *discouraged by* **RS652**

Related to P.2.

651 Ruleset
R652 *and* **R508**

RS651 *permits* **R704** *JK*

653 Rule (*JK* **,** *OIL* **)**
It is obligatory for the conference system to support electronic reception of papers.

R653 *obligated by* **R652**
R653 *obligated by* **R659**
R653 *necessitated by* **R703**

654 Rule (*JK* **)**
It is recommended for the OC to only support one paper format.

R654 *recommended by* **R6**

655 Rule (*JK* **)**
It is obligatory for the OC to support ordinary mail.

R655 *obligated by* **R17**

656 Rule (*JK* **)**
For the use of electronic communication the stakeholders must have internet access.

R656 *obligated by* **RS505**

657 Rule (*JK* **)**
It is obligatory for the OC to be able to receive papers on paper.

R657 *obligated by* **R12**
R657 *obligated by* **R18**

Related to P2.

658 Rule (*JK* **)**
It is discouraged to scan in scientific papers.

652 Ruleset
R658 *and* **R657** *and* **R654**

RS652 *discourages* **R652**

659 Rule (*JK* **)**
It is obligatory for contributers to issue electronic papers in a standard format.

R659 *obligates* **R653** *JK*
R659 *obligates* **RS653** *JK*
R659 *obligates* **R660** *JK*

R659 *recommended by* **R652**

660 Rule (*JK* **)**
It is obligatory for the OC to state clearly the formats of electronic submission it support.

R660 *obligates* **R602**

R660 *obligated by* **R659**

661 Rule (*JK* **)**
It is obligatory for the OC to support postscript.

662 Rule (*AS* **)**
It is recommended for the OC to support RTF.

PRUNED.

653 Ruleset
R661 *or* **R662**

RS653 *obligated by* **R659**

663 Rule (*JK* **)**
It is obligatory for the conference system to keep track of the current state of all papers.

R663 *obligated by* **R502**

Related to E18 Rel20 E21.

Reviews :

701 Rule (*JK* **,** *AS* **)**
It is obligatory for the conference system to support the reviewing of papers.

R701 *obligated by* **R202**

Related to P4.

702 Rule (*JK* **)**
It is obligatory to be able to send out papers on paper for review.

R702 *obligated by* **R20**

Related to P3.3.

703 Rule (*JK* **)**
It is recommended to be able to send out electronic papers for review.

R703 *necessitates* **R653**

R703 *recommended by* **R508**

Related to P3.3.

704 Rule (*JK* **)**
It is recommended to be able to receive electronic reviews.

R704 *obligates* **R705** *AS*
R704 *recommends* **R708** *JK*
R704 *obligates* **R707** *JK*

R704 *recommended by* **R508**
R704 *permitted by* **RS651**

701 Ruleset
R221 *and* **R704**

RS701 *obligates* **R709** *JK*

Related to P4.

705 Rule (*JK* **)**
It is recommended to be able to receive reviews on email.

R705 *recommends* **R707** *JK*

R705 *obligated by* **R704**

Related to P4.

706 Rule (*JK* **)**
It is obligatory for the conference system to supply manual review-forms.

R706 *obligated by* **R20**

Related to P3.3.

707 Rule (*JK* **)**
It is obligatory for the conference system to supply electronic review-forms.

R707 *obligated by* **R704**
R707 *recommended by* **R705**

Related to P4.

708 Rule (*JK* **)**
It is recommended for the conference system to support reception of reviews over WWW.

R708 *recommended by* **R704**

Related to P4.

709 Rule (*JK* **)**
It is permitted for PC-members to issue electronic reviews.

R709 *obligated by* **RS701**

Related to P4.

702 Ruleset
R708 *and* **R709**

710 Rule (*JK* **)**
It is obligatory for the conference system to create reminder list over missing reviews.

R710 *recommends* **R711** *JK*
R710 *obligates* **R712** *AS*
R710 *obligates* **R713** *JK*

711 Rule (*JK* **)**
When two weeks before review deadline it is obligatory for the conference system to create a general reminder to those PC-members who have not yet given in all their reviews.

R711 *recommended by* **R710**

Related to R4.7.

712 Rule (*JK* **)**
When past the review deadline it is obligatory for the conference system to create reminders to those PC-members who have not given in their reviews.

R712 *obligated by* **R710**

Related to R4.8.

713 Rule (*JK* **)**
If the review is entered by WWW, it is obligatory that an identification check is performed on the reviewer.

R713 *obligated by* **R156**
R713 *recommended by* **R220**
R713 *obligated by* **R221**
R713 *obligated by* **R710**

Related to P4.1.

714 Rule (*JK* **)**
If before review-deadline then it is permitted for a PC-member to update an existing review over WWW.

Related to P4.2.

715 Rule (*JK* **)**
If before review-deadline then it is permitted for a PC-member to delete an existing review over WWW.

Related to P4.2.

716 Rule (*JK* **)**
It is obligatory for the conference system to support customized reviewforms.

R716 *recommended by* **R504**

Related to E13, E14, E15, E16, E17 and relationships involving these entity-classes.

PC-meetings and sessions :

751 Rule (*AS* **)**
It is obligatory for the conference system to support selection of papers at the PC-meeting.

Related to P5.1.

751 Ruleset
R251 *and* **R751**

RS751 *recommends* **R752** *AS* **A** It is often more convenient for many of the program committee members to attend a meeting a special place different from the site of the conference e.g. in connection with another conference.

752 Rule (*AS* **)**
It is obligatory for the conference system to be portable, so that at PC-meeting can be held and supported by the system anywhere.

R752 *obligates* **R506** *AS* **A** We have portable PCs available, but not portable UNIX-machines.

R752 *recommended by* **RS751**

753 Rule (*AS* **)**
It is obligatory for the conference system to supply a report sorted according to the best results as the average of all criteria.

R753 *recommended by* **R253**

PRUNED.

754 Rule (*AS* **)**
It is obligatory for the conference system to supply a report sorted according to different criteria.

R754 *obligated by* **R253**

Related to P5.2.1.

755 Rule (*AS* **)**
It is obligatory for the conference system to supply a report including the review-details for each single paper.

R755 *obligated by* **R253**

Related to P5.2.4.

756 Rule (*AS* **)**
It is obligatory for the conference system to supply a report including the official review-details only for each single paper.

R756 *obligated by* **R218**
R756 *obligated by* **RS251**

Related to P5.2.3.

757 Rule (*AS* **)**
It is recommended to be able to put weights on different categories.

R757 *recommended by* **R202**

Related to Att21.

758 Rule (*AS* **)**
It is recommended to be able to put weights on different reviewers.

R758 *recommended by* **R202**

Related to Att26.

759 Rule (*AS* **)**
It is recommended for the conference system to keep track of sessions, papers assigned to sessions, and potential conflicts.

R759 *recommended by* **R502**

PRUNED.

CRC :

801 Rule (*JK* **)**
It is obligatory for the conference system to keep track of conditional accepts.

R801 *obligated by* **RS501**

Related to P6.2. and E21.

802 Rule (*JK* **)**
It is obligatory that the conference system can create reminder-list over CRCs.

R802 *obligated by* **RS502**

Related to P6.6.

Participation :

851 Rule (*AS* **)**
It is recommended to build up a database of places to advertise the conference.

R851 *recommended by* **R351**

PRUNED.

852 Rule (*JK* **)**
It is recommended to make the conference program available on WWW.

R852 *obligated by* **R351**

Practical organization :

901 Rule (*OIL* **)**
It is obligatory for the conference system to support the organization of the ISDO95 conference.

R901 *recommended by* **R502**

PRUNED

902 Rule (*OIL* **)**
It is obligatory for the organizing chair to have good cooperation and coordination with SEVU.

R902 *obligates* **R903** *OIL*
R902 *recommends* **R904** *OIL*
R902 *recommends* **R905** *OIL*

R902 *obligated by* **R401**

903 Rule (*OIL* **)**
It is obligatory to communicate with SEVU using standard-schemas.

R903 *obligated by* **R902**

904 Rule (*OIL* **)**
It is recommended to communicate with SEVU by sending data in electronic form.

R904 *recommended by* **R508**
R904 *recommended by* **R902**

PRUNED.

905 Rule (*OIL* **)**
It is recommended to communicate with SEVU by directly updating their database.

R905 *discouraged by* **R21**
R905 *recommended by* **R508**
R905 *recommended by* **R902**

PRUNED.

Proceedings :

951 Rule (*AS* **)**
It is recommended for the conference system to support the electronical distribution of the proceedings.

R951 *forbidden by* **R456**
R951 *recommended by* **R508**

Related to P7.1.

951 Ruleset
R452 *and* **R951**

RS951 *obligates* **R953** *AS*

952 Rule (*AS* **)**
It is obligatory to issue a paper proceedings.

R952 *recommended by* **R22**
R952 *obligated by* **R456**

953 Rule (*AS* **)**
It is obligatory to support electronic sale of the proceedings.

R953 *obligated by* **RS951**

PRUNED.

Project Management :

1001 Rule (*AS* **)**
It is recommended for the project group to create a generally applicable conference system.

R1001 *obligated by* **RS1001**

1002 Rule (*GR* **)**
It is recommended for the group to get a good grade on the project.

R1002 *obligates* **R1003** *AS*

1003 Rule (*GR* **)**
It is recommended that the group make customer is satisfied with their work.

R1003 *obligated by* **R1002**

1001 Ruleset
R504 *and* **R1003**

RS1001 *obligates* **R1001** *AS*

1004 Rule (*AS* **)**
It is obligatory for the group to finish the project within 1/12-94.

R1004 *obligates* **RS504** *JK*

1005 Rule (*GR* **)**
It is recommended for the group to keep within the time-budget.

In Figure A.47, an example is given on the links from high-level rules to rule Rof process P8.1.

A.2 Tempora-cases

We have in the thesis used some examples from the Sweden Post case-study and the library case originally used in Tempora. Even if we have only been loosely associated with Tempora, the experience from this project haveirectly inspired several of our suggestions regarding the use of rules in conceptual modeling, especially when it comes to the intergrationn rules and other conceptual modeling languages, the use of deontic operators, and the use of inter-rule operators. During the project, we participated in meetings in Trondheim, London, and Stockholm, some of which included modeling of the Sweden Post case study. In addition we have had long discussions with those deeply involved in the case study, and have had access to all the material produced as a result of the project. The work reported in [179, 197, 198] was the result of work that we were involved in that also benefited and influenced Tempora.

The library case study was originally performed by Løvseth based on the design documentation of an operational library system [203] using the Tempora languages. Parts of this case havelso been used by us to investigate the use of deontic operators [167], and the link between the rule-language of Tempora and PPM and PLD of PPP [168, 179].

The descriptions of the cases below are based on the descriptions given by Seltveit in [282].

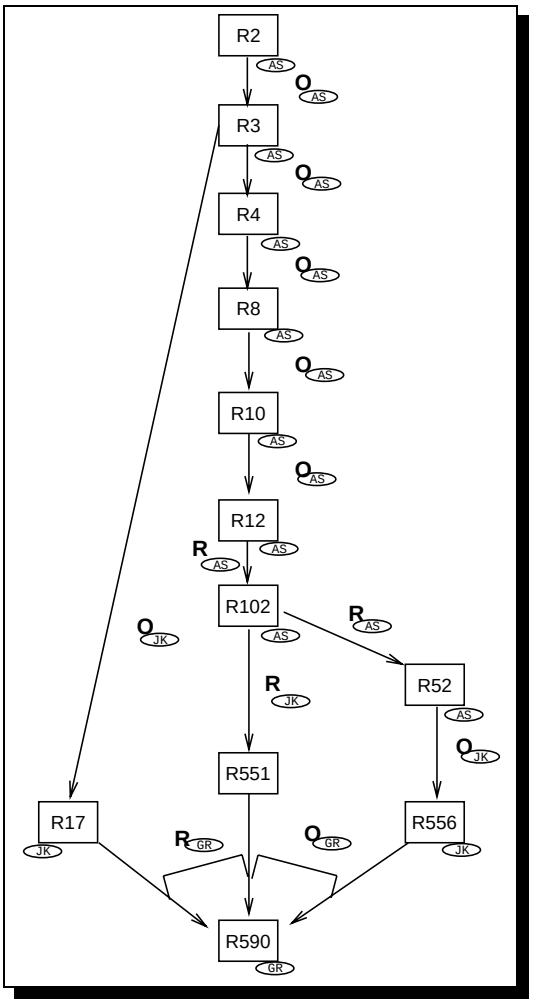

Figure A.47: Example of rule hierarchy built bottom-up

A.2.1 Sweden Post

The Post is a large governmental held company responsible for the transportation and delivery of mail and packages nationwide in Sweden. The subject of the case study was the order and invoicing system for the mail delivering part of the Post. This aims at supporting the Post's business with large customers, i.e. customers that send large quantities of mail.

The Post offers a wide range of articles and each article has a normal price. However, the normal price can be over-ruled by agreements. An agreement is customer specific. It specifies what discounts the customer should be given and the conditions that must be fulfilled by the customer, e.g. the minimum quantity of mail and how it should be paid.

Each mail delivery from a customer is received together with a number of delivery notes and one covering letter. A delivery note specifies the details about the articles in the delivery. The covering letter contains general data about the delivery such as number of delivery notes, number of delivery note lines for the various notes, and serial number of the registration post office.

To get paid for its articles, the Post produces a claim to the customer based on the data in the delivery notes and in the agreements. A claim to the customer gives raise to an invoice which is sent to the customers for payment. One invoice may specify articles from one or more mail

deliveries.

Based on the data provided from delivery notes and the Post's internal accounts, various types of statistics are produced. For instance, sales statistics are requested by sales and marketing departments at different Post's regions.

Some of the models created in connection with this case has been presented in Chapter 7.2.

A.2.2 The Library Case

A library information system aims at supporting all the major activities at the library, such as:

- Acquisition, consisting of ordering , assessment, and cataloging. A document (e.g. a book or a report) is considered for purchase if it is suggested by a client or a librarian, e.g. after review of brochures from librarians or by shortage of copies of popular documents. Suggestions for acquiring documents are compiled in acquisition lists. The lists are checked to see if the document exist and compared with the library budget. Decisions are made whether a document should be ordered for inspection, for purchase, or not at all. Finally orders are sent to the library's regular vendors according to its rules of purchase.
- Administration, including a number of activities such as registration of clients, decision about acquisition of documents, and management of loaned documents.
- Public services, include general information services and assisting clients that want to borrow or return documents. Anyone whether registered or not can use the general information services, e.g. accessing the library databases from public terminals. Documents may be copied as regulated by Norwegian law.

Figure B.1: The PPP repository structure

Appendix C

DRL Syntax

The syntax of the extended rule language is described using a modified Backus-Naur form rules, with [square] brackets indicating optional sections, and { braces } optional repeating sections. Terminals are written *emphasized* whereas non-terminals are written in ordinary text. Some of the more detailed restrictions on the syntax are not included. This applies especially to the restrictions on the form of triggers and consequences.

Inter-rule relationships

Ruleset	::=	DRL_rule | DRL_rule or Ruleset | DRL_rule and Ruleset
Deonconn	::=	Ruleset Deonrel Ruleset *based on* Argument
Binconn	::=	DRL_rule Binrel DRL_rule
Deonrel	::=	*obligates* | *recommends* | *permits* | *discourages* | *forbids* | *excludes*
Binrel	::=	*necessitates* | *causes* | *uses* | *has_info_derived_by* | *overrules* | *suspends*
Argument	::=	String

DRL rules

DRL_rule	::=	[*when* exp][*if* exp]*then*[*it is* deonop [*for* role]] exp [*else* [*it is* deonop [*for* role]] exp]
deonop	::=	*obligatory* | *recommended* | *permitted* | *discouraged* | *forbidden*

Note that this is somewhat simplified. See Chapter 9.2 for a further specification and difference between the 'exp' in the different parts of the rule.

Access and Selection of an ONER Scenario

exp	::=	ONER_access
ONER_access	::=	ONER_entityclass_instance | ONER_entityclass_instance ONER_addaccess | ONER_entityclass_instance *[* ONER_addaccess {ONER_addaccess } *]*
ONER_addaccess	::=	ONER_relationship ONER_access | ONER_relationship *has* ONER_attribute | ONER_relationship *has* ONER_method | *has* ONER_attribute | *has* ONER_method

ONER_relationship ::= ONER_id_name | = | < | > | =< | >= | <>
ONER_entityclass_instance ::= ONER_entityclass | string | time_ point | arithmetic_exp
role ::= ONER_access | string
ONER_entityclass ::= ONER_id_name [. atomic_element]
ONER_id_name ::= small_alpha{ alpha | _}
ONER_attribute ::= ONER_id_name | set |
ONER_id_name *has* ONER_attribute | ONER_id_name ONER_access
ONER_method ::= ONER_id_name *(* { ONER_attribute } *)*
atomic_element ::= string | integer | float | blob | boolean | variable | time_point
element ::= atomic_element | tuple | set | interval
element_list ::= element { ,element}
structure ::= tuple | set | variable
arithmetic_exp ::= integer | float | variable | arithmetic_exp arithmetic_binop
arithmetic_exp | arithmetic_unop arithmetic_exp | time_unit / time_unit
| *(* arithmetic_exp *)* | *number_of* structure | *minimum* structure |
maximum structure | *average* structure
string ::= "{ ASCII char} "
integer ::= digit { digit }
float ::= integer . integer
boolean ::= *true* | *false*
arithmetic_binop ::= + | - | * | / | *mod*
arithmetic_unop ::= + | -
capital_alpha ::= *A* | *B* | . . . | *Z*
small_alpha ::= *a* | *b* | . . . | *z*
alpha ::= capital_alpha| small_alpha
digit ::= *1* | *2* | . . . | *9*

Quantification, set expressions and tuples

exp ::= *for_all* ONER_access *it_follows_that* exp | *exists* ONER_access *such_that* exp
exp ::= structure == structure | element *member_of* structure
set_variable_list ::= variable{, variable} | tuple
set_binop ::= *union* | *intersect* | *disjoint*
set ::= { set_variable_list *for which* exp} | [ONER_id_name]{ [element_list]}
| *setof (* structure, exp*)* | *(*set*)* | set set_binop set
tuple ::= { set_variable_list *for_which* exp} |
[ONER_id_name]{ element_list | structure } |
bagof(structure, exp*) (*tuple*)* | *project(* tuple,tuple,structure*)*
exp ::= ONER_id_name *(*[element_list]*)*

Logical and temporal operators and connectives

exp ::= *true* | *false*
exp ::= exp logical_binop exp | *not* exp | *only_ one_of* exp | *(*exp*)*
logical_binop ::= *and* | *or*
exp ::= temporal_unop exp | exp temporal_binop exp

temporal_unop ::= *at_any_time* | *sometime_in_ past* | *sometime_in_future* |
always_in_past| *always_in_future* | *just_before* | *just_after*
temporal_binop ::= *until* | *since*
exp ::= exp interval_test interval | interval interval_ relationship interval
interval ::= *[* time_point, time_point *]* | interval >>time_unit|interval<<time_unit
| time_point | variable | date | interval_of *(* exp *)* | *today* | *yesterday*
| *tomorrow* | *this_week* | *next_week* | *this_month* | *next_month*
| *this_year* | *last_year* | *next_year*
interval_test ::= *during* | *for*
interval_relationship ::= *runs_into* | *coends* | *contains* | *beginning_of* | *equals* | *ends_before*
| *ends_at_start_of* | *ends_during* | *ends_with* | *ends_after* | *starts_before*
| *starts_with* | *starts_during* | *starts_at_end_of* | *starts_after*

Time points: Arithmetic manipulation of time_unit involving ground terms may be checked for semantic correctness. Time_points can refer to both absolute (11::11::1911) and relative (start_of_today) time.

exp ::= *time_is* time_point | *time_is* time | exp *at* time_point | exp *after* time_unit
| exp *before* time_unit | *day_is* day_name | *week_ number_is* week_name.
time_point ::= time date | time | variable | time_point >> time_unit | time_point << time_unit
time_point_of (exp*)* | *start_of_today* | *start_of_yesterday*| *start_of_tomorrow* |
start_of_this_week | *start_of_last_week* | *start_of_next_week* | *start_of_this_month*|
start_of_last_month | *start_of_next_month* |
start_of_this_year | *start_of_last_year* | *start_of_next_year* |*end_of_today*|
end_of_yesterday| *end_of_tomorrow* | *end_of_this_week* | *end_of_last_week* |
end_of_next_week | *end_of_this_month*| *end_of_last_month* | *end_of_next_month* |
end_of_this_year | *end_of_last_year* | *end_of_next_year*
time ::= hour *:* minute *:* second
date ::= day *::* month *::* year
time_unit ::= time | variable | integer | float | *second* | *minute* | *hour* | *day* | *week*
seconds | *minutes* | *hours* | *days* | *weeks*| time_unit + time_unit
time_unit - time_unit | time_unit * time_unit
time_unit * arithmetic_exp | time_unit / arithmetic_exp
day_name ::= variable | *Monday* | *Tuesday* | *Wednesday* | *Thursday* | *Friday* | *Saturday* | *Sunday*
week_name ::= variable | *00* | *01*| . . . |*53*
second ::= variable | *00* | *01*| . . . |*59*
minute ::= variable | *00* | *01*| . . . |*59*
hour ::= variable | *00* | *01*| . . . |*23*
day ::= variable | *00* | *01*| . . . |*31*
month ::= variable | *00* | *01*| . . . |*12*
year ::= variable | *1900* | *01*| . . . |*9999*

Appendix D

Integration of the Conceptual Extensions and PPP Modeling Techniques

We present in this appendix ideas for how to extend the following modeling techniques taking the extensions of the conceptual framework into account:

- Extending the general execution facilities.
- Extending the explanation generation facilities.
- Extending the filtering mechanisms to include rule-hierarchies and actor models.
- Extending the general tool-support.

We will in the following only look upon how the existing techniques as described in existing work can be extended with actors, roles, and rules, applying the same basis that has already been used there. We will not discuss improvements of these techniques beyond what is needed to incorporate our extensions. Thus we will only indicate the feasibility of supporting the new phenomena and not if the total approach is necessarily the best one, taken all modeling languages into account.

D.1 General Execution

We will in this section sketch how the general execution mechanisms described in [335] can be extended to take actors, roles, and rules into account. Since we have not extended ONER with temporal expressiveness as in Tempora ERT, we will not look into detail on how to support an underlying historical database, even if the rule-language as it is specified in Appendix C can be used for querying a historical database. [305] contains an overview of how querying and updating of a historical database can be performed efficiently.

To illustrate how the general execution mechanism can be extended, we first describe the general framework in which execution is taking place, i.e. the meta-model. The meta-model is written in a language combining a data modeling language with a language for specification of context-free grammars. This data-modeling language is illustrated in Figure D.1.

This modeling language and the accompanying meta-model is different from the other meta-models made in connection with PPP. One could easily switch to using ONER, but we have not done this below to make it easier to compare with the existing work. The meta-model can also represent the executional aspects of other conceptual modeling languages than the ones given in PPP.

The extended meta-model is shown in Figure D.2. We will explain the complete model below, and emphasize the additions made by us. For the additional terms the explanation follows [335].

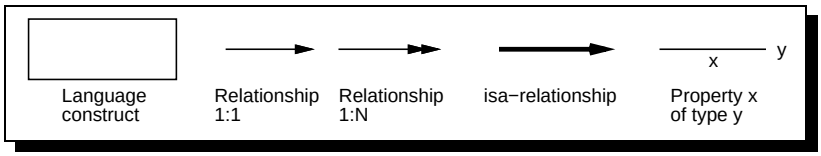

Figure D.1: Data modeling language for meta-modeling (from [335])

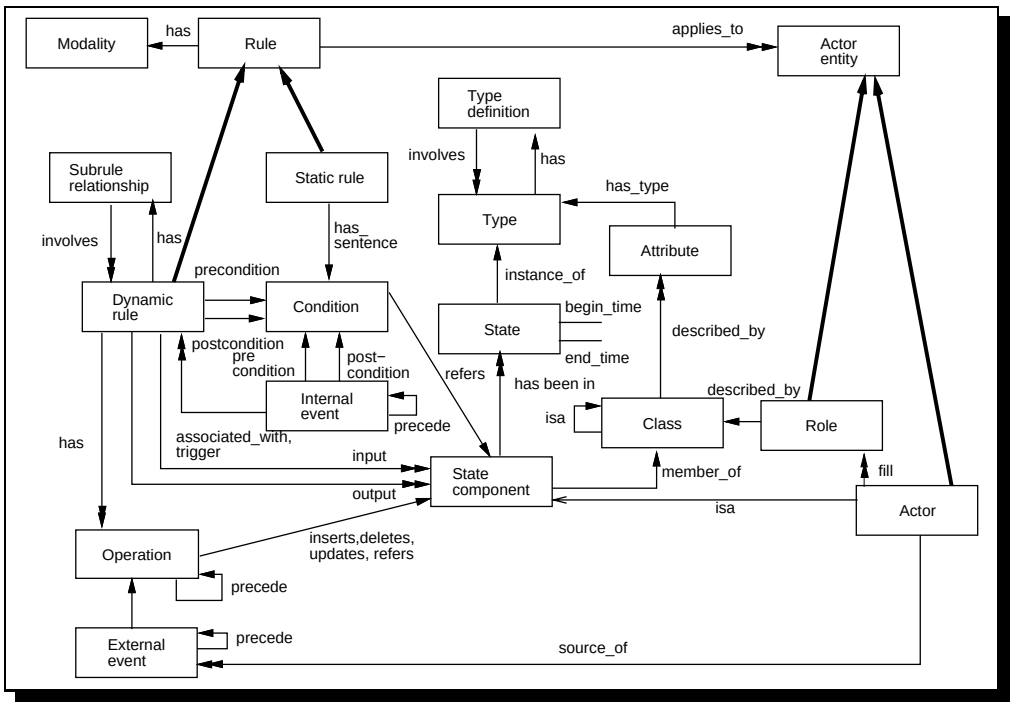

Figure D.2: Meta-model for execution in PPP

- State component: Something that can be uniquely identified, and has an associated state. Excluding executional aspects, the state of a system is the aggregate state of all state components in the system. State components correspond to entities, the static part of objects, *actors,* named variables etc.
- Class: A class has a name, and consists of a set of state components which have common properties.
- Attribute: A function which maps a state component to a value. The value of all properties of a state component in a class constitute the state of that state component.
- Type: An intentional description of state components. It has a name and a definition.
- Type definition: Specifies a type in terms of other types through type constructor like sets, aggregates etc.
- Condition: A logical sentence which refers to states of state components and is either true or false in a given state.
- Static rule: Includes a condition which should hold in all states. It further has a name, it specifies the situation in which the condition should be evaluated, and it specifies what action should be taken if it is violated.
- *State*: As defined above, but including in addition a begin-time and an end-time of the state.

This is a first step towards supporting an underlying historical database.

- *Role*: Generally, behavior that can be expected by an actor by other actors. Here it is the static description of the role which is represented.
- Dynamic rule: Causes state transitions in a pre-described manner. It covers construct like process, rule, activity, method, etc.
- Sub-rule relationship: Links dynamic rules in a hierarchical rule structure, i.e. it specifies control structures between dynamic rules. It has a name, and it must have a specification.
- Operation: A specification of a state change or a query. It may insert, delete, or update state components.
- External event: A state change imposed by the environment of the system. The state change is related to two subsequent system states.
- Internal event: A state change imposed by a dynamic rule in the system.
- *Actor:* The initiator of external events.
- *Actor entity:* A generalization of role and actor.
- *Rule:* A generalization of state and dynamic rules.
- *Modality*: The modal status of a rule. Possible values are necessitation, obligation, recommendation, permission, discouragement, prohibition, and exclusion.

The relationships in the meta-model are further explained in Table D.1.

Based on this meta-model a language ECML has been developed. This is mostly adapted as defined by Willumsen, but the definitions of STATIC_RULE and DYNAMIC_RULE have been extended as indicated below. Only the main parts of the language are presented here.

- CLASS(CLASSNAME,ATTRIBUTE(ATTRIBUTENAME,TYPENAME)):
  Several attributes can be indicated for a class.
- TYPE(TYPENAME,TYPEDEF):
  Defining complex types.
- ISA(CLASS, CLASS):
  Used to specify generalization.
- VARIABLES(TYPENAME,VARIABLENAME):
  Indicates that a variable is of a given type.
- STATIC_RULE(ID,OPERATION,STATECOMPONENT,SENTENCE,ACTION,MODALITY):
  Here MODALITY has been added to the original format. OPERATION is one of insert, delete, and update. STATECOMPONENT is a path specifying which state components that are involved. SENTENCE is a condition in a subset of first order logic, similar to in the relational calculus. This could be extended to first order temporal logic to be able to support a historical database. The ACTION is a routine to be called in case the static rule is violated.
- QUERY(LIST,FORMULA).
- INSERT(CLASSNAME).
- DELETE(FORMULA,LIST).
- UPDATE(FORMULA, SET OF ASSIGNMENTS).
  An assignment further consists of two parts, a STATE COMPONENT and a FUNCTIONAL EXPRESSION.

Relationship	Explanation
applies_to(*r,a*)	A rule *r* applies to a set of roles and actors *a.*
associated_with(*ie,dl*)	Event *ie* is caused by the execution of *dl.*
deletes(*o,sc*)	Operation *o* deletes a state component *sc* from a class.
described_by(*c,a*)	All members of class *c* have attribute *a.*
described_by(*r,c*)	A role *r* is described by a class *c.*
fill(*a,r*)	Actor *a* fill role *r.*
has_type(*a,t*)	The value of *a* is in the extension of *t.*
has(*t,td*)	Type *t* is defined through type definition *td.*
has(*dr,sr*)	The complex dynamic rule *dr* has subrules in relationship *sr.*
has(*dr,o*)	Dynamic rule *dr* is simple, and *o* executes as part of *dr.*
has(*r,m*)	A rule *r* has modality *m.*
has_been_in(*sc,s*)	State component *sc* has been or are currently in state *s.*
has_sentence(*sr,c*)	Static rule *sr* requires *c* to hold in every state.
input(*dr,sc*)	State component *s* is an input parameter to *dr.*
inserts(*o,s*)	Operation *o* inserts a state component *s* into a class.
instance_of(*s,t*)	The state *s* is in the extension of *t.*
involves(*td,t*)	Type definition *td* involves *t.*
involves(*sr,dr*)	Sub-rule relationship *sr* involves sub-rule *dr.*
isa(*c1,c2*)	All member of class *c1* are members of class *c2.*
member_of(*sc,c*)	State component *sc* is a member of class *c.*
output(*dr,s*)	State component *s* is an output parameter from *dr.*
precede(*o1,o2*)	Operation *o1* is executed before operation *o2*
precede(*ee1,ee2*)	External event *ee1* must happen before *ee2.*
precede(*ie1,ie2*)	External event *ie1* must happen before *ie2.*
precondition(*dr,c*)	Condition *c* must hold for *dr* to apply.
precondition(*ie,c*)	Condition *c* holds in the state when *ie* happens.
postcondition(*ie,c*)	Condition *c* holds in the state when *ie* has happened.
postcondition(*dr,c*)	Condition *c* must hold in the state *dr* terminates.
refers(*o,sc*)	Operation *o* refers to a state component *sc* in a class.
refers(*c,sc*)	Condition *c* involves reference to a state component *sc.*
source_of(*a,ee*)	The source of an external event *ee* is actor *a.*
specified_by(*ee,o*)	Operation *o* is performed when *ee* is reported.
trigger(*ie,dr*)	The event *ie* may trigger the execution of *dr.*
updates(*o,sc*)	Operation *o* updates a state component *sc* in a class.

Table D.1: Relationships in meta-model for general execution

- DYNAMIC_RULE(ID, PRECONDITION, BODY, POSTCONDITION, INPUT, OUTPUT, PROPERTIES, ACTION, MODALITY).
  Here MODALITY and ACTION have been added to the original format. ACTION indicate the action to be taken when a PRECONDITION evaluates to false. The default value for MODALITY is obligatory. BODY is either a list of operations, or a sub-rule-relationship in the form SUBRULE(SUB-RULE RELATIONSHIP,LIST OF SUB-RULES).
  The following sub-rule relationships are used in connection with the execution of PPP-models:
  - VCS: Vague control structure. Executes sub-rules in a spontaneous manner when their preconditions hold.
  - SEQ: Indicate a temporally ordered sequence of sub-rules.
  - EXCSEQ: If the precondition of several dynamic rules applies, only the first in the sequence is executed.
  - REPSEQ: Repeats a sequential execution of sub-rules if the precondition still holds after that the last rule is executed.
- EXTERNAL_EVENT(TRANSACTION,OPERATIONS([INSERT(TRANSACTION)]))

In addition it is possible to indicate standard operators such as ADD, MULT, SUB, DIV, MIN, MAX, NOT, AND, XOR, IMPLIES.

To incorporate a historical database as in Tempora, queries (and also inserts, deletes, and updates) should be able to refer to previous and not only the current state. We will not go into detail about this here, but start with laying the groundwork for how time and deontic operators can be supported.

D.1.1 Incorporating Time

As indicated in the semantics for the prototype, also to deal with the deontic operators, it is necessary to have time explicitly represented. The following needs to be added to take these aspects into account, using a similar semantics as in Section 9.3.2. The main idea here is to support the simulation the two time-scales as discussed in Section 9.3.2.

Before executing a model, one must indicate:

- GRANULARITY: This is how many seconds that are taking place per tick.
- MODEL_SECONDS: This is how many model-seconds the model should execute.
  MODEL_PERIOD = MODEL_SECONDS / GRANULARITY.
- REAL_SECONDS: How many clock-seconds the model should execute.
  REAL_PERIOD = REAL_SECONDS / GRANULARITY.

REALVSMODEL = REAL_PERIOD / MODEL_PERIOD.

At the start of an execution we have:

- MODEL_TIME = 0.
- START_TIME = SYSTEM_TIME [1].

The following two rules are included to update the clocks:

- To update MODEL_TIME:
  DYNAMIC_RULE(MODEL_CLOCK,MODEL_TIME $<$ MODEL_PERIOD,
  OPERATIONS([UPDATE(TRUE,[(MODEL_TIME,ADD(MODEL_TIME,DIV(1,GRANULARITY)))])]),
  TRUE, [], [], [], [], NECESSITATION)

[1] Taken from the system-clock.

Note that model_time can also be used in connection to indicate begin and end_times of states.

- To update REAL_TIME:
  DYNAMIC_RULE(REAL_CLOCK,REAL_TIME < START_TIME + REAL_ PERIOD,
  OPERATIONS([UPDATE(TRUE,[(REAL_TIME, MAX(SYSTEM_ TIME,
  ADD(START_TIME,TIMES(MODEL_ TIME,REALVSMODEL)))])]), TRUE, [], [], [], [], NECESSITATION)

In this way it is possible to simulate two time-scales, and this can be used to implement the borrowed time semantics.

In addition we need the following dynamic rule:

DYNAMIC_RULE(CLOCK, TRUE,SUBRULES(SEQ,[SYSTEM_RULE,MODEL_ CLOCK,REAL_ CLOCK]), TRUE, [], [], [], [], NECESSITATION)

The SYSTEM_RULE has all rules corresponding to processes in a top level PPM as sub-rules.

D.1.2 Actors and Roles

The static aspects of roles are represented as classes, based on the entity-classes they are linked to. Roles and sub-roles are linked through a traditional isa-construct.

The static part of actors is represented as a type, similarly to other state-components, indicating the data about the actor.

For actors filling roles, one can represent this as:

ISA(ACTORNAME,CLASSNAME)

which is also represented recursively through the role-network by default.

D.1.3 Processes and Rules

Flows from role/actors to and from processes and timers are represented as classes:

CLASS(FLOW,ATTRIBUTE(ATTRIBUTE,TYPENAME))

The type is defined through the type-construct, and contains an initiator of type actor in addition to the type of the contents of the flow.

For a timer that has a flow entering it, a similar class is specified. Flows between actors are not represented, whereas additional flows are represented in other ways as described in [335].

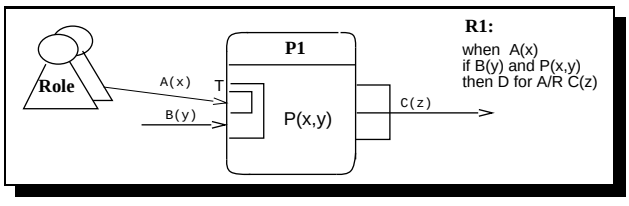

Figure D.3: Execution of deontic rules

Based on Figure D.3 have we the following dynamic rules:

For a single rule R1, we use the following shorthand:

DYNAMIC_RULE(R1,A(INITIATOR,X),B(Y) AND P(X,Y) AND DELETE(A),C(INITIATOR,Z), [], [], [], [], D)

INITIATOR is the initiator of the rule. How it is applied in connection with deontic operators is illustrated below. Note that the forwarding of the initiator also must be done in the dynamic rules made to represent the receive and send constructs of PLD. B(Y) AND P(X,Y) must be translated into operators, or alternatively, sub-rules.

For the different deontic operators, we have the following:

- D = Necessary (i.e. blank) or obligated. No additions, but the modality is indicated in MODALITY.
- D = Recommended: The following additional precondition to the dynamic rule is given:
  REAL_TIME $\leq$ SYSTEM_TIME
  In addition can the following ACTION be specified (which will be run in case the rule that is executed do not fire):
  RECOMMENDATION_OVERRULED
- D = Permitted: In this case the treatment will be dependant of if the rule applies to a role or to an actor:
  - Role: Add in PRECONDITION: ISA(INITIATOR,ROLE)
  - Actor: Add in PRECONDITION: INITIATOR=ACTOR

  The above is the default. In the case that the rule is specifically overruled on the actor level, the precondition based on the role is extended with AND INITIATOR $\neq$ ACTOR1 ...
- D = Discouraged: The same preconditions as for permission are specified, in addition do one have REPORT_DISCOURAGED as part of BODY.
- D = forbidden: Again will we have two cases according to if the rule applies to an actor or a role:
  - Role: Add in PRECONDITION: NOT ISA(INITIATOR,ROLE)
  - Actor: Add in PRECONDITION: NOT INITIATOR = ACTOR

  Add as ACTION: SECURITY_VIOLATION(INITIATOR).
  The above is the default. In the case that the rule is specifically overruled on actor level, the precondition for the role is extended with OR INITIATOR = ACTOR1 ...

When there are more than one rule linked to a process, the representation is similar as it is for a decomposed process, i.e.

DYNAMIC_RULE(PID,PRECONDITION,SUBRULES(VCS,RULES), TRUE, [], [], [], [],OBLIGATION)

PRECONDITION is made based on the port structure of the process.

D.1.4 Deontic Constraints

The constraint "If A then modality B" can be supported using a static rule:

- STATIC_RULE(ID,OPERATIONS,STATECOMP,SENTENCE,INCONSISTENCY, MODALITY)

where OPERATIONS, STATECOMP, and SENTENCE must be generated based on the constraint.

When it comes to handling deontic and other inconsistency, this should be handled as part of the general execution-cycle. Willumsen have not treated this in detail. Proper inconsistency-handling and rollback mechanisms is a large research area in itself that will be regarded as outside the scope of the thesis.

D.1.5 Timers

Generally, the TIMER_PERIOD indicated in real or relative time is translated to ticks according to the GRANULARITY as indicated above.

We differentiate between clocks and delays, and different types of these.

- Clocks:
  - A clock with one outflow, no inflows or offflows:
    On start of the system: Next_signal = number ticks to first outflow.
    - DYNAMIC_RULE(TIMER1, NEXT_SIGNAL $>$ 0, OPERATIONS([UPDATE(TRUE,[(NEXT_SIGNAL,SUB(NEXT_SIGNAL,1))])]),TRUE, [], [], [], [], OBLIGATION).
    - DYNAMIC_RULE(TIMER2, NEXT_SIGNAL $=$ 0, OPERATIONS([INSERT(OUTFLOW)]), OPERATIONS([UPDATE(TRUE,[(NEXT_SIGNAL,TIMER_PERIOD)])], [], [], [], [], OBLIGATION)

    In addition there is a dynamic rule.
    - DYNAMIC_RULE(TIMER,TRUE,SUBRULES(EXCSEQ,[TIMER1,TIMER2]), TRUE, [], [], [], [], NECESSITATION)

    which again is part of the VCS structure of the system_rule. If there are several outflows, this is added to the operation above.
    OUTFLOW is specified as a class, where the class name is the name of the flow, and the type is taken from the flow-specification similarly as with normal flows. In addition is the ATTRIBUTE(SYSTEM,INITIATOR) added as the first attribute.
    If there are several outflows, there are several insert-operations in the body of the second rule.
  - Both inflow and outflow, but no offflow:
    In this case, NEXT_SIGNAL is set to -1 on start. The main rule is:
    - DYNAMIC_RULE(TIMERIN,INFLOW,TRANSFER-DATA-AND-REMOVE-INFLOW, OPERATIONS([UPDATE(TRUE,[(NEXT_SIGNAL,TIMER_PERIOD)])], [], [], [], [], OBLIGATION))

    TRANSFER-DATA-AND-REMOVE-INFLOW is a shorthand for putting the data that has come on the inflow including the initiator in temporary storage. In addition must the item on the inflow be removed. This is similar to the treatment of the PLD receive construct.
    The DYNAMIC_RULE for the outflow is the same as the one above
    In addition there is a dynamic rule:
    - DYNAMIC_RULE(TIMER,TRUE,SUBRULES(EXCSEQ,[TIMER1,TIMER2,TIMERIN]), TRUE, [], [], [], [], NECESSITATION)

    This indicates that when a clock has received an onflow, it will not react to other inflows, since the timer can not be turned off. This is not truly general, and could be remedied by adding the counter to the class to represent the timer, and store the different counters here.
    When several inflows exist, several rules of the type TIMERIN is also created. These are collected in a rule of the form:
    - DYNAMIC_RULE(TIMERIN,TRUE,SUBRULES(VCS,[TIMERIN1,TIMERIN2,...]), TRUE, [], [], [], [], NECESSITATION)

    The super-rule of this is equal to the EXCSEQ-rule above.

If the timer-period is based on the inflow, the update of NEXT_SIGNAL is based on the item on the inflow, and not the TIMER_PERIOD directly.

– Both inflow, outflow, and offflow are modelled:
  The following rule is added based on the offflow:
  * DYNAMIC_RULE(TIMEROFF,OFFLOW,REMOVE-TEMPORARY-AND-OFFLOW-DATA, OPERATIONS([UPDATE(TRUE,[(NEXT_SIGNAL,-1)])], [], [], [], [], OBLIGATION).

  The body REMOVE-INFLOW-AND-OFFLOW-DATA remove the item that was supposed to be sent on the outflow, and in addition remove the offflow item.
  The additional super-rule
  * DYNAMIC_RULE(TIMERALL,TRUE,SUBRULES(EXCSEQ,[TIMEROFF,TIMER]), TRUE, [], [], [], [], NECESSITATION).

  is added to stop a timer before it triggers.
  If there are several offflows, these all have a separate dynamic_rule, and these are collected in a VCS structure similar to what was done in the case of several onflows.

- Delays:
  For a delay, the situation is parallel to the situation with a clock with both onflow and offflow, with the change that NEXT_SIGNAL is set to -1 when the data is forwarded, if not already removed by an offflow.

D.2 Explanation Generation

To illustrate how to improve the explanation generation facilities in detail, we need to give a more comprehensive overview of the mechanisms behind this. We will then illustrated how to represent the additional language features by utilizing the overall mechanisms as described in [114].

The explanation generation process is divided into two subprocesses:

- Deep generation: The content and structure of the explanation are determined, creating a deep explanation. This is independent of the language in which the final explanation is to be presented.
- Surface generation: Produce a surface explanation consistent with a deep one in a given (set of) language(s).

Ideally these processes are independent. In practice some interaction oftentakes place. Other components includes the source model, a set of discourse strategies, a lexicon, and a grammar (see Figure D.4.) The source model is a model generated especially for the purpose from the conceptual model. A user model might also be included as mentioned in Chapter 7.4, but this is not described here.

We will basically describe the changes needed due to the extension in the conceptual modeling languages on the level of the source model, i.e. the mapping from the conceptual modeling languages to the source model. Also necessary extensions to the discourse strategies are outlined. We do not cover extensions needed for surface generation.

The explanation generation is in general based on two linguistic theories, Functional Grammar (FG) and Rhetorical Structure Theory (RST). We apply these theories in the similar manner as is done by Gulla [114].

The source model is meant to be independent of the conceptual modeling languages in the CASE environment, thus, what Gulla terms a domain structure model (DSM) is applied. This contains two kinds of elements, phenomena and relationships between phenomena, and is in a

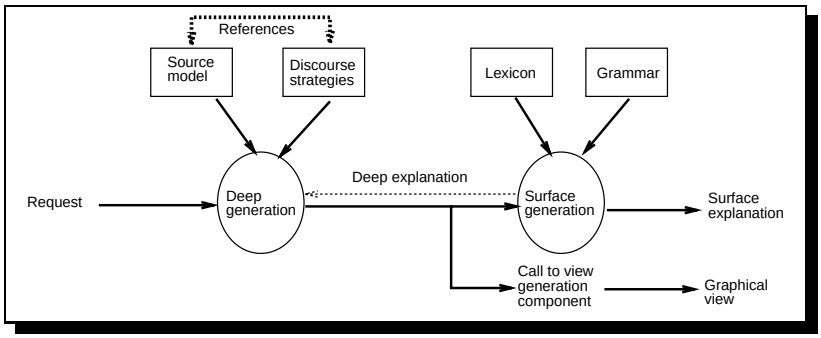

Figure D.4: Steps and representations in explanation generation

sense a language model described using parts of ONER. The defined phenomena in DSM are:

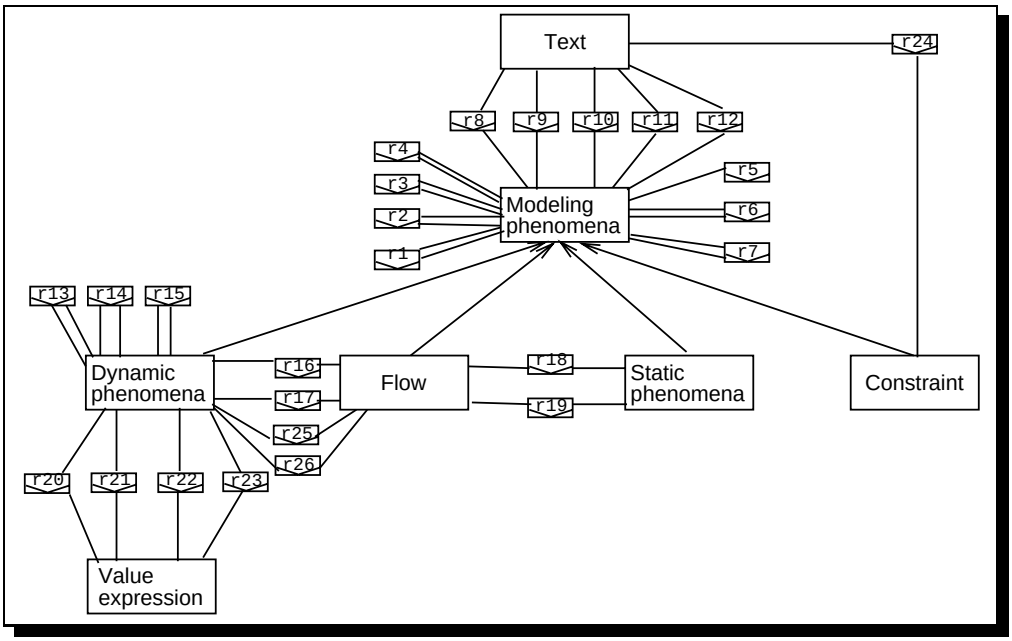

Figure D.5: Domain structure model

- **Modeling_phenomenon(E):** This is the most general phenomenon in DSM. It represents any phenomenon E found in the modeling languages. Specializations of phenomenon are dynamic_phenomenon, static_phenomenon, flow, and constraints.
- **Text(E):** E is a text in natural language.
- **Dynamic_phenomenon(E):** A dynamic phenomenon has some kind of behavior. It can be triggered by other dynamic phenomena, or it can be activated as other dynamic phenomena terminate. It can receive flows and can have precondition for execution, and its results may involve generated flows and preconditions. It can be deactivated by terminating signals from other dynamic phenomena, or on the basis of a specified duration.
- **Flow(E):** A flow can be either a data flow or a control flow. It has an origin, which provides the flow, and a destination that consumes or keeps it for later use. Both dynamic and static phenomena may serve as origins and destinations.

- **Value_expression (E):** Any expression involving elements of conceptual models. It may be assignments or tests, but can just as well be more complicated e.g. SQL-queries.
- **Static_phenomena(E):** A static phenomena E is any structure found in a conceptual model that can be counted as static with respect to what is being modeled.
- **Constraint(E)** An element E is a constraint if it is used to specify constraints on the conceptual model.

The relations in DSM relates instances of phenomenon to each other as illustrated in Figure D.5. The relationships are described in Table D.2 and are partly deduced from similar RST-relations. The small changes that we have done to the original model are similarly based on RST.

No	Relationship	Description
r1	equivalence(E_1,E_2)	E_1 is equivalent to E_2.
r2	possession(E_1,E_2)	E_1 can be said to possess E_2 from some point of view.
r3	extension(E_1,E_2)	E_2 belong to E_1's extension .
r4	generalization(E_1,E_2)	E_1 is a generalization of E_2.
r5	aggregation(E_1,E_2)	E_1 is an aggregation that includes E_2 .
r6	classification(E_1,E_2)	E_2 is an element of E_1.
r7	relationship(E_1,E_2)	E_2 is involved in relationship E_1.
r8	characterization(E_1,E_2)	E_1 has a textual characterization E_2.
r9	denotation(E_1,E_2)	E_1's purpose is described textually as E_2.
r10	justification(E_1,E_2)	The use of E_1 is justified by E_2.
r11	motivation(E_1,E_2)	The use of E_1 is motivated by E_2.
r12	rationale (E_1,E_2)	The existence of E_1 is justified by E_2.
r13	trigger(E_1,E_2)	Dynamic phenomenon E_1 activates E_2.
r14	terminate(E_1,E_2)	Dynamic phenomenon E_1 terminates E_2.
r15	succeed(E_1,E_2)	The termination of dynamic phenomenon E_1 initiate the execution of phenomenon E_2.
r16	receive(E_1,E_2)	Dynamic phenomenon E_1 receives flow E_2.
r17	generate(E_1,E_2)	Dynamic phenomenon E_1 generates flow E_2.
r18	get (E_1,E_2)	Static phenomenon E_1 gets flow E_2.
r19	provide(E_1,E_2)	Static phenomenon E_1 provides flow E_2.
r20	precondition(E_1,E_2)	E_2 is a precondition for dynamic phenomenon E_1.
r21	postcondition(E_1,E_2)	E_2 is a postcondition for dynamic phenomenon E_1.
r22	duration(E_1,E_2)	Dynamic phenomenon E_1 has duration given by E_2.
r23	act(E_1,E_2)	Dynamic phenomenon E_1 performs E_2.
r24	syntax(E_1,E_2)	Constraint E_1 has syntactic description E_2.
r25	start (E_1,E_2)	Arrival of flow E_1 activates dynamic phenomenon E_2.
r26	stop(E_1,E_2)	Arrival of flow E_1 terminates dynamic phenomenon E_2.

Table D.2: Relationships in model for explanation generation

From this model, one can generate sentences in the explanation modeling language EML as illustrated in [114], which are used in the actual generation of the deep explanation. We will below illustrate how the different new situations that appears because of our conceptual extensions can be represented in DSM. How the existing PPP-languages are represented is illustrated in [114] and will not be repeated here.

D.2.1 Actors and Roles

The DSM-specification of roles and actors based on Figure D.6 are as follows

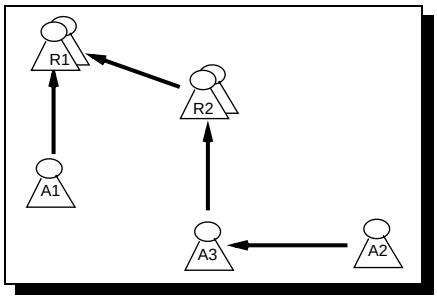

Figure D.6: Representation of actors and roles

Actors are represented as dynamic phenomena, whereas roles are represented as static phenomena. This, including the relationships, are represented by the following relations:

- DYNAMIC_PHENOMENON(A1),DYNAMIC_PHENOMENON(A2),DYNAMIC_PHENOMENON(A3)
- STATIC_PHENOMENON(R1),STATIC_PHENOMENON(R2)
- CLASSIFICATION(R1,A1),CLASSIFICATION(R2,A3)
- GENERALIZATION(R2,R1). This also implicitly implies CLASSIFICATION(R2,A1)
- AGGREGATION(A3,A2)

D.2.2 Processes and Rules

How explanation generation for rules in Tempora can be represented is discussed briefly in [114], and the treatment of the simple case follows a similar pattern.

The simple situation with one rule describing a process is depicted in Figure D.7.

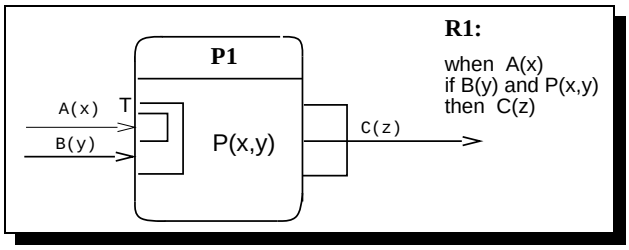

Figure D.7: Integration of process and rule in PPP

A possible representation in DSM are:

- 1. DYNAMIC_PHENOMENON(P1)

- 2. START(A,P1)
- 3. RECEIVE(P1,B)
- 4. GENERATE(P1,C)
- 5. AGGREGATION(P1,R1)
- 6. DYNAMIC_PHENOMENON(R1)
- 7. START(A(X),R1)
- 8. PRECONDITION(R1,B(Y) $\wedge$ P(X,Y))
- 9. POSTCONDITION(R1,C(Z))

1-4 above deals with the representation of the process, 5 links the rule and the process, and 6-9 represents the rule. Alternatively 8 above can be represented as two clauses:

- 1. PRECONDITION(R1,B(Y))
- 2. ACT (R1,P(X,Y))

This might result in explanations that are easier to comprehend. When there are several rules describing a process, this will be represented by duplicating 5-9 above for all rules.

The detailed contents of a rule is not dealt with in this scheme. There are two possible ways of attacking this: Either to paraphrase the formal rule, in which case this is to be taken care of by the surface generator, or to use the accompanying quasi-natural language form of the rule if it is developed. In this case, one should assure that the formulations are equal.

When including deontic operators and actors the following extensions are needed referring back to Figure D.3.

A/R indicates either the role or the specific actor the rule is set to apply to. We will discuss the two cases in more detail below. Also the cases for the separate deontic operators will be discussed, referring to the semantics of the deontic operators described in Chapter 9.3.2.

Similarly to above, the basic situation is illustrated with the following:

- 1. DYNAMIC_PHENOMENON(P1)
- 2. START(A,P1)
- 3. RECEIVE(P1,B)
- 4. GENERATE(P1,C)
- 5. AGGREGATION(P1,R1)
- 6. DYNAMIC_PHENOMENON(R1)
- 7. START(A(X),R1)
- 8. PRECONDITION(R1,B(Y) $\wedge$ P(X,Y))
- 9. POSTCONDITION(R1,C(Z))
- 10. STATIC_PHENOMENON(ROLE)
- 11. PROVIDE(ROLE,A(X))

10 and 11 are new compared to the situation illustrated above. The real time (t_R) and the time in the temporal module(t_M) is assumed to be available.

In a given case, there will be an actor A1 initiating the situation, which is dynamically represented as

- DYNAMIC_PHENOMENON(A1)
- GENERATE(A1,A(X))
- POSTCONDITION(P1,GENERATE(A1,C(Z)))

Whereas the second item handles the triggering of process P1, indicating who is the source of the trigger, the last item is to be able to transfer the initiation data further through the process network.

For the different possible operators, we have the following:

- D = Necessary (i.e. blank) or obligated. No additional relationships are necessary.
- D = Recommended: An extra precondition to the rule is stated

PRECONDITION(R1,$t_M \leq t_R$)
In addition is the following process specified:
  - DYNAMIC_PHENOMENON(RECOMMENDATION_OVERRULED)
  - PRECONDITION(RECOMMENDATION_OVERRULED,$t_M > t_R$)
  - PRECONDITION(RECOMMENDATION_OVERRULED,B(Y) $\wedge$ P(X,Y))
  - PRECONDITION(RECOMMENDATION_OVERRULED,GENERATE(_,A(X)))
- D = Permitted: In this case the treatment will be different if the rule applies to a role or an actor:
  - Role: PRECONDITION(R1,CLASSIFICATION(A1,R))
  - Actor: PRECONDITION(R1,EQUIVALENCE(A,A1))
- D = Discouraged: The same preconditions as above for permission are specified, in addition do one have POSTCONDITION(R1,REPORT_DISCOURAGED)
- D = Forbidden: Also here we will have two cases according to if the rule applies to an actor or a role:
  - Role: PRECONDITION(R1,$\neg$ CLASSIFICATION(A1,R))
  - Actor: PRECONDITION(R1,$\neg$ EQUIVALENCE(A,A1))

  For permissions and prohibitions, a similar pattern as for recommendation_overruled can be used to indicate security_violations, but is not shown here. The precondition can also be more complex as indicated for the general execution.

D.2.3 Rule Hierarchies

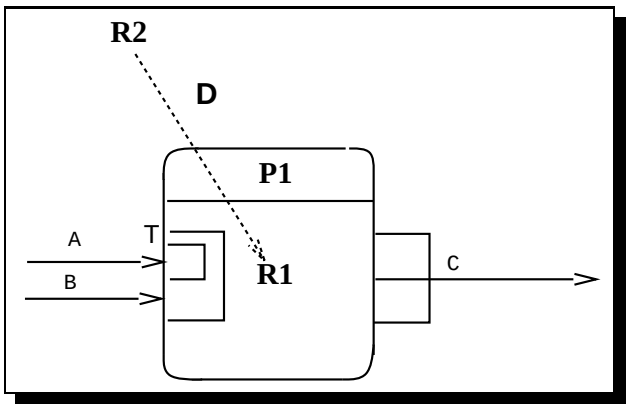

Figure D.8: Explanation generation based on rule-relationships

A simple example applying rule hierarchies is illustrated in Figure D.8 where **R2** is in relationship with the **R1** here describing the process. **R1** is represented as illustrated above, whereas for **R2**, the representation will be dependant on the relationship D as illustrated below:

- D = *Necessitates*: RATIONALE(R1,R2), RATIONALE(P1,R2).
- D = *Obligates*: JUSTIFICATION(R1,R3), JUSTIFICATION(P1,R2).
- D = *Recommends*: MOTIVATION(R1,R2), MOTIVATION(P1,R2).

The other rule-relationships are not of interest from the point of view explanation generation. The new 'motivation' relationship indicates that the discourse strategy for justification must be updated to take this into account. The underlying structure currently deals with the standard deontic operators of obligated, permitted, and forbidden (= not permitted). To extend with recommenda-

tion and discouragement, the discourse strategies including deontic operators generally needs to be updated so that the deontic modality is given as a parameter to the strategy. The changes to the given explanation is syntactical only, and can be dealt with in surface generation. This area especially applies to normal deontic rules above. The actual justification will be made based on the rule, applying a quasi-natural form or a paraphrasing of a formal form as described above.

As an example, we have used the situation taken from the conference example depicted in Figure D.9.

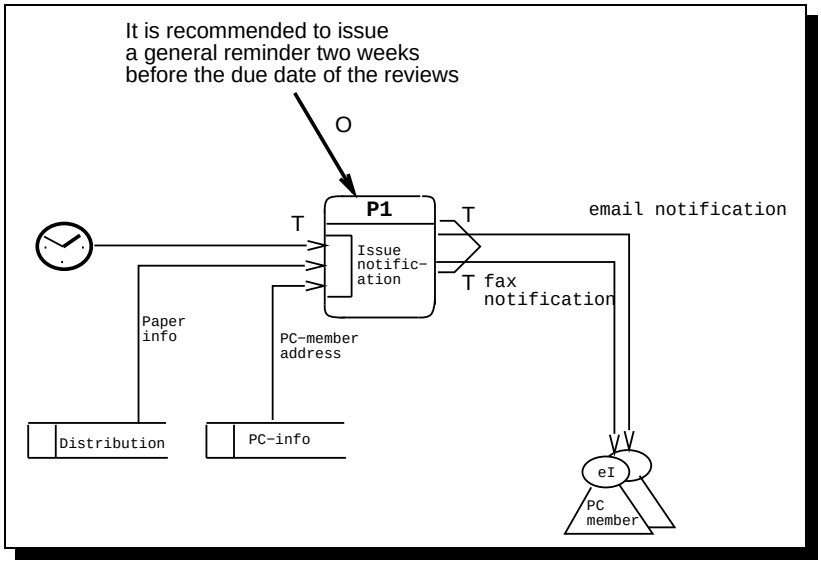

Figure D.9: Scenario for explanation

The request 'justification(p1)' would produce a structure depicted in Figure D.10.

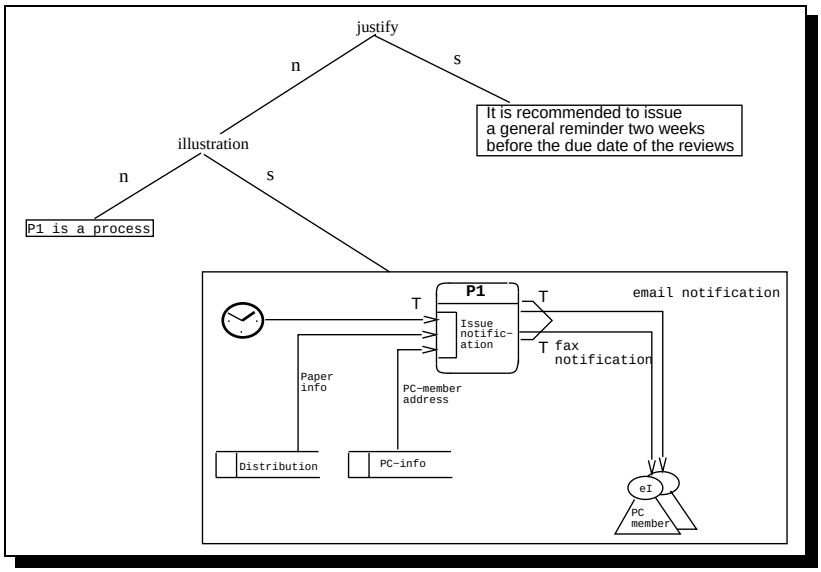

Figure D.10: Explanation structure

In the case of several rules being related to rule R1, this is dealt with in the following way, referring to Figure 9.9:

- **R1** *recommends* **R3**, **R2** *obligates* **R3**: This translates into MOTIVATION(R3,R1), JUSTIFICATION(R3,R2) with similar relationships between the rules and the process as illustrated above.
- **R1** or **R2** *obligates* **R3**. Since these kind of situations will have to be dealt with (i.e. only one of **R1** and **R2** being active at a time) this situation will be similar to the simple situation described above.
- **R1** and **R2** *obligates* **R3**: **R1** and **R2** will be represented as a rule-set **RS1**, and this is represented with
  - AGGREGATION(RS1,R1)
  - AGGREGATION(RS1,R2)
  - JUSTIFICATION(R3,RS1)

  If **R2** is institutionalized by actor A1, this is represented with:

  POSSESSION(A1,R2)

If R2 *overrules* R1, this is including by adding an extra precondition to R1:
PRECONDITION(R1,$\neg$ B2), where B2 is the condition of R2. Similarly can be done for *suspend.*

D.2.4 Deontic Constraints

This can be supported using CONSTRAINTS, and adding deontic operators as parameters to the paraphrase and correction strategy.

D.2.5 Timers

Because of the extended forms of rules that can be linked to timers, we also defined the DSM-representation for these. We differentiate between clocks and delays.

- Clocks: Also here, it is necessary to differentiate between different cases:
  - A clock with one outflow, no inflow or offflow:
    - DYNAMIC_PHENOMENON(TIMER)
    - TRIGGER(SYSTEM START,TIMER)
    - DURATION(TIMER,TEMPEXP)
      Here TEMPEXP is based on the temporal expression in the rule for timer.
    - POSTCONDITION(TIMER,OUTFLOW)
    - START(OUTFLOW,TIMER)

    If there are several outflows, several postconditions are specified, one for each outflow.
  - Both inflow and outflow, but no offflow:
    In this case, the trigger above is substituted with START(INFLOW,TIMER). When several inflows exist, several triggers are specified, one for each inflow.
  - Both inflow, outflow and offflow is present:
    STOP(OFFLOW,TIMER) is added to the set of relations.
- Delays:
  - GENERATE(A,ONFLOW): Based on the actor who started the overall process.
  - START(ONFLOW,TIMER)

- DURATION(TIMER,TEMPEXP)
  TEMPEXP can be determined dynamical based on the contents in on-flow.
- POSTCONDITION(TIMER,GENERATE(A,OUTFLOW))
- STOP(OFFLOW,TIMER)

D.3 Filtering

The filtering mechanisms suggested by Seltveit for Tempora [282] should be adapted to PPP and extended. When it comes to the inclusion of rules, similar filters as suggested for Tempora should be included, and extended based on the inclusion of roles and deontic operators. In addition filters for actor models and links between models should be included.

We will here give an overall presentation of all filters in PPP, in a similar way as has been done by Seltveit. We will partly use a similar formalization and the same notation for easier comparison:

A filter $\mathcal{F}$ is a mapping from a specification that contains all possible viewspecs V^* to a viewspec V.

The viewspec can again be divided according to the languages used, thus for PPP we have:

$$\mathcal{F} : \{V^*_{ONER}, V^*_{PPM}, V^*_{DRL}, V^*_{Rulerel}, V^*_{actor}, V^*_{Link}\} \rightarrow \{V_{ONER}, V_{PPM}, V_{DRL}, V_{Rulerel}, V_{actor}, V_{Link}\}$$

where

- $V^*_{ONER} =< O_1, O_2, O_3, O_4, O_5, O_6 >$ and
  $O_1 = \{Entity_classes\}, Entity_classes = Entity_name1, Entity_name2, \ldots,$
  $O_2 = \{Relationship_classes\},$
  $O_3 = \{Generalizations\},$
  $O_4 = \{Types\},$
  $O_5 = \{Methods\},$
  $O_6 = \{Cardinality_constraints\},$
- $V^*_{PPM} =< P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10} >$ and
  $P_1 = \{Processes\},$
  $P_2 = \{Stores\},$
  $P_3 = \{Roles\},$
  $P_4 = \{Timers\},$
  $P_5 = \{Flows\},$
  $P_6 = \{Ports\},$
  $P_7 = \{Actors\},$
  $P_8 = \{Sinks\},$
  $P_9 = \{Speech_acts\},$
  $P_{10} = \{Supports_relationships\},$

- $V^*_{DRL} =< R, classification, institutionalizer >, < R >= \{< \tau, \phi, \psi, \rho, \mathbf{D} >\}$
  where this indicates the elements of the rule and the type of the rule as it is classified. The institutionalizer is the actor who has institutionalized this rule if any.
- $V^*_{Rulerel} =< R_1, R_2, R_3, R_4 >$ and
  $R_1 = \{Rulesets\},$
  $R_2 = \{Rule_relationships\},$

$R_3 = \{Node_types\}$,
$R_4 = \{Arguments\}$,

- $V^*_{actor} =< A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9 >$ and
  $A_1 = \{Actors\}$,
  $A_2 = \{Roles\}$,
  $A_3 = \{Agents\}$,
  $A_4 = \{Relationships\}$,
  $A_5 = \{Support_relationships\}$,
  $A_6 = \{Interactions\}$,
  $A_7 = \{Power_relationship\}$,
  $A_8 = \{in_role\}$,
  $A_9 = \{on_behalf_of\}$,

- $V^*_{Link} =< L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8, L_9, L_{10}, L_{11} >$ and

  $L_1 = \{Process_rules\}$,
  $L_2 = \{Timer_rules\}$,
  $L_3 = \{Actor_rules\}$,
  $L_4 = \{Role_rules\}$,
  $L_5 = \{Applies_to\}$,
  $L_6 = \{Sources\}$,
  $L_7 = \{Agrees\}$,
  $L_8 = \{Flow_scenarios\}$,
  $L_9 = \{Actor_scenarios\}$,
  $L_{10} = \{Rule_scenarios\}$,
  $L_{11} = \{Access\}$,

- $V \subseteq V^*$

$V \subseteq V^*$ means

- $V_{ONER} =< O'_1, O'_2, O'_3, O'_4, O'_5, O'_6 >$

- $V_{PPM} =< P'_1, P'_2, P'_3, P'_4, P'_5, P'_6, P'_7, P'_8, P'_9, P'_{10} >$
- $V_{DRL} =< R', classification', institutionalizer' >, < R' >= \{< \tau', \phi', \psi', \rho', \mathbf{D} >\}$,
- $V_{Rulerel} =< R'_1, R'_2, R'_3, R'_4 >$
- $V_{actor} =< A'_1, A'_2, A'_3, A'_4, A'_5, A'_6, A'_7, A'_8, A'_9 >$
- $V_{Link} =< L'_1, L'_2, L'_3, L'_4, L'_5, L'_6, L'_7, L'_8, L'_9, L'_{10}, L'_{11} >$

where for all the model elements of the model, there are subset related pairs of model elements (K, K') such that K is a model element of the originating model, and K' is a corresponding model element in the generated viewspec.

In the rest of this section, we will outline a set of filters which can be used for modeling with the PPP-languages. The choice of a particular set of filters is obviously highly subjective. The proposal is based on the filters proposed for the Tempora-languages, which are based on experience gained through case-studies applying the Tempora and PPP-languages, and experience and requirements put forward by other researchers and practitioners on the field. This is discussed in more detail by Seltveit.

In addition to the classification of filters as language and model filters other relevant aspects

of filters include [282]:

- Inclusiveness/exclusiveness.
- Determinism/non-determinism.
- Global/local effects.
- Projection/approximation.

With the exception of the first item below, we will only describe the filters informally. A further formalization could follow the same pattern as in [282]. Filters can be combined freely. It should be possible to store combinations of filters as macros to be used at a later stage.

D.3.1 ONER Filters

The following filters are specified for ONER-models:

- Entity_class:
  Inclusive: Remove all model elements from a model except for entity-classes, and generalization relationships between entity-classes.
  This can be formally expressed as:

  $$\mathcal{F}_{entity_class}: V^*_{ONER} \rightarrow V_{entity_class}$$

  where
  - $V^*_{ONER} =$
    $< Entity_classes, Relationship_classes, Generalizations,$
    $Types, Methods, Cardinality_constraints >$
  - $V_{entity_class} =< Entity_classes, \emptyset, Generalizations, \emptyset, \emptyset, \emptyset >$
- Generalization:
  Inclusive: All entities that have sub or super-classes are including, with the addition of the generalization relationships.
  Exclusive: All generalization-relationships are removed.
- Types:
  Exclusive: Remove all types and attributes.
  Inclusive: Include only types which are immediately linked to an entity-class, relationship-class, or method, i.e. ignoring sub-types.
- Methods:
  Exclusive: All methods, and links from methods to entity-classes, relationship-classes, and types are removed.
- Cardinality-constraints:
  Exclusive: Removes all cardinality-constraints.
- Model element-filter:
  Select an entity-class, relationship-class, or method, and include all other model elements that can be reached by N links. N is by default 1. If the model element that is reached is a relationship class, this also include the other entity-classes linked to the relationship-class. Based on the selection of an entity-class, all relationship-classes, attributes, and methods that are attached to this entity-class or one of its super-classes are included.

D.3.2 PPM Filters

The following filters are specified for PPM-models:

- Process overview:
  Inclusive: Give an overview picture in a tree notation of all processes including their decompositions, linking sub and super-processes.
- Store:
  Exclusive: Remove all stores from a PPM, including the flows entering or leaving the store, and support-relationships. If this results in that the role or actor that supported the store is no longer linked to any other model element, this is also removed.
- Roles:
  Exclusive: Remove all roles from a PPM, including the flows entering or leaving the role, and the support relationships involving the role.
- Timers:
  Exclusive: Remove all timers from a PPM, including the flows entering or leaving the timers.
- Flows:
  Exclusive: Collapsing of flows with the same input or output or both should be supported as described in [282].
- Control flow:
  Inclusive: Only triggering flows and the model elements they link are retained.
  Inclusive: Both triggering and terminating flows and the model elements they link are retained.
- Ports:
  Partial exclusive: The outermost post remains, whereas the inner ports are removed.
  Total exclusive: All port-structures are removed. This can often be followed with a restructuring of the diagram.
- Actors:
  Exclusive: Remove all actors from a PPM, including the flows entering or leaving the actor, and the support relationships involving the actor.
- Sinks:
  Exclusive: Remove all sinks from a PPM, including the flows entering the sink.
- Speech acts:
  Exclusive: All speech-act indication on flows are removed.
  Inclusive: All flows that are linked to a conversation with a propositional content that is given are included, together with the model elements that these flows leave or enter.
- Support:
  Exclusive: Remove all support links. Also remove actors or roles that have no flows entering or leaving them.
  Inclusive: Include all model elements that are supported by a specified actor or role.
- Model elements:
  Select a process, an actor, a role, a store, or a timer, and include other model elements that can be reached by N links. N is by default 1.
  Path filter: Select a process and include all model elements on all paths of the process by following all output-flows of the process. (The functionality of this filter can also be achieved by applying the above component filter several times, but in might be convenient to be able to do this using few commands).

D.3.3 Rule Filters

For a set of rules, the following filters are defined:

- Operator:
  Inclusive: Include all rules with a given deontic operator.
  Exclusive: Remove all rules with a given deontic operator.
- Actor:
  Inclusive: Include all rules that applies directly to the given actor.
- Role:
  Inclusive: Include all rules that applies directly to a given role.
- Classification:
  Inclusive: Include all rules of a given type.
- Institutionalizer:
  Inclusive: Include the set of rules that is institutionalized by a given actor.

D.3.4 Rule Relationship Filters

The following filters are specified for rule hierarchies:

- Rule-relationships:
  Inclusive: Include all relationships that have a given type, together with the rule-set they link.
  Inclusive subset: Based on one rule, include all rules linked to this rule recursively in both directions, and include also all the linking relationships.
  Exclusive: Remove rule-relations of a given type, and rule-sets that end up without links after removing the rule-relationships.
- Argument:
  Exclusive: Remove the arguments from a rule-hierarchy
- Node-type:
  Inclusive: Include the nodes of a given node-type, e.g. all and-nodes.
- Issue:
  Inclusive: Include rules involved in an issue, as described i Section 9.2.
- Model element:
  Select a rule, and include all rules that are linked with this rule in both direction through N links. The default of N is 1. One should also be able to indicate if one wants to include pruned branches in the rule hierarchy.

D.3.5 Actor Model Filters

For actor models, the following filters are suggested:

- Actors:
  Inclusive: Include all actors and links between actors.
  Inclusive types: Include all actors of a specific type as indicated in Chapter 9.1 and links between these actors.
  Inclusive part: Include all actors that are either internal or external to a given actor, and all links between these actors.
- Roles:
  Inclusive: Include all roles and links between these roles.
  Inclusive types: Include all roles of a specific type and links between these roles.
  Inclusive part: Include all roles that are either internal or external to a given actor, and all links between these roles.

- Agents:
  Exclusive: Remove all agents and links to agents. Do not remove the links between the actor and role in the agent.
- Part_of:
  Inclusive: Include all actors and roles linked through a part_of link.
- Fill:
  Inclusive: Include all actors and roles linked through fill links.
- Institute:
  Inclusive: Include all actors and roles linked through an institute-link.
- Support:
  Inclusive: Include all support relationships, and the actors that supports each other.
  Exclusive: Remove Suppoert-relationships from a diagram.
- Power:
  Inclusive: Include all power relationships, and the actors that supports each other.
  Exclusive: Remove power-relationships from a diagram.
- Interaction:
  Inclusive: Include all potential communication paths, and actors communicating with each other.
  Exclusive: Remove actor-communication from a diagram.
- In_role:
  Exclusive: Remove the in_role indication on links.
- On_behalf_of:
  Exclusive: Remove the on_behalf_of indication on links.
- Model element:
  Select an actor or a role, and include all other actors and roles that are linked through N links, also including the links. N is by default 1.

D.3.6 Filters Combining Models in Several Languages

The following "filters" utilize the link between different models. They are not filters in the usual sense, but are useful for supporting the browsing between different models.

- For a process: List all rules that applies to the process. For a process that is decomposed, this includes also the rules linked to the decomposed processes.
- For a timer: List all rules that applies to a timer.
- For an actor or role: List all the rules that are used to simulate the behavior of the actor.
- For an actor: List all rules applying to an actor, also inheriting the rules applying for the roles the actor currently fill.
- For a role: List all rules that apply to a role and its sub-roles.
- For a process: Based on the link from the flows entering and leaving the process, return the scenario the process have on the ONER-model.
- For a store: Based on the link from the flows entering and leaving the store, return the scenario the store have on the ONER-model.
- For a rule: Return the scenario the rule have on the ONER-model.
- For a flow: Return the scenario the flow-description have on the ONER-model.
- For an actor: Return the scenario of the ONER-model that the actor has access to.
- For an actor or role: Return the scenario of the ONER-model that describes the structural aspects of the actor.

- For a set of actors: Indicate all interaction between a set of actors.
- For a scenario to the ONER-model: List all rules that applies to some of the model elements in the scenario.
- Source:
  Inclusive: Include statements which the indicated actor is the source of.
- Agree:
  Inclusive: Include statements which the indicated actor agrees to.

D.4 Additional Tool Support

We will in this section outline some of the additional tool-support needed in the PPP-tool because of the extensions made in this thesis. The overview is based on the existing tool as it is described in [150] and on descriptions in [4, 196, 282, 340]. We will in particular concentrate on actor-modeling, but will for completeness outline the general functionality. Editors for rules and rule-hierarchies are discussed in [259, 282].

We will concentrate the description of the modeling tool, and not on general setup and administrative and project management issues. We will also indicate the inclusion of additional modeling functionality as discussed in Chapter 6 which is not yet addressed in detail in PPP.

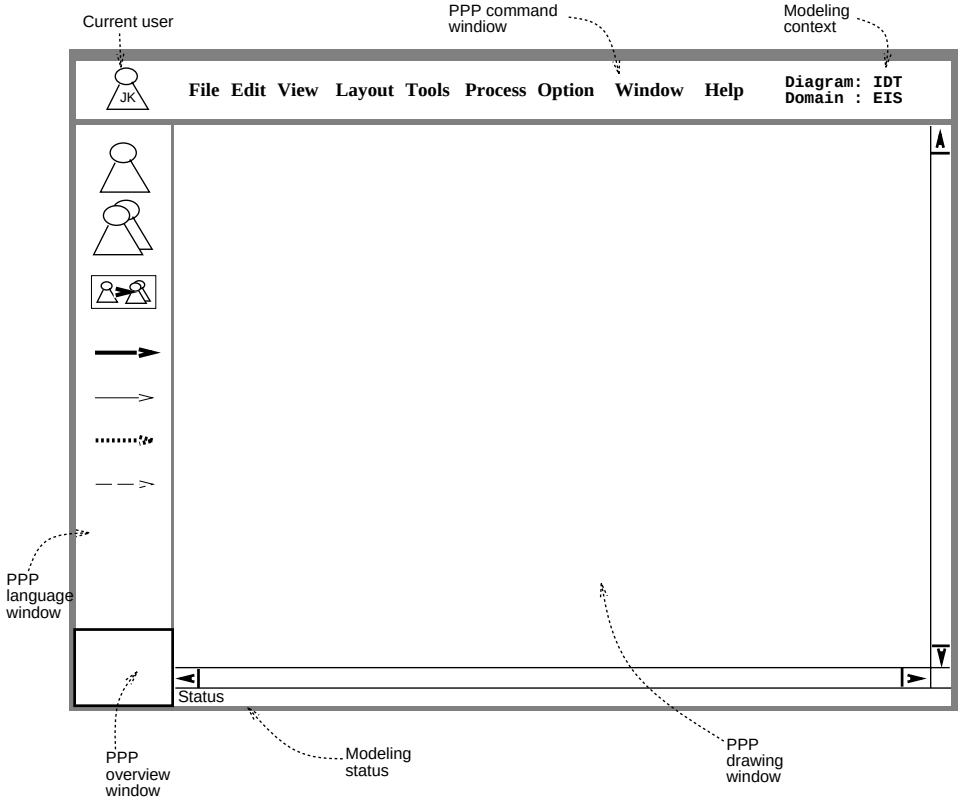

Figure D.11: Example of tool-layout

Figure D.11 gives a general picture of a potential modeling tool for actor modeling. In addition the inclusion of different icons and speed-keys can be useful, but these are not pictured.

The window consists of four sub-windows. The PPP command window, The PPP language window, The PPP drawing window, and the PPP overview-window. The command window shows the major functions for manipulating models, the language window shows the main nodes and links between nodes that can be use in modeling, whereas the drawing window is used for the actual modeling. The overview window shows the total model, whereas the drawing window only shows parts of this according to the zooming of the window. In addition, the window contains indications of overall modeling context, current user, and modeling status.

Basic functionality, such as selecting a symbol, resizing a symbol and moving a symbol is directly available. The same is the selection of a node-symbol in the palette and putting this into the diagram, and selecting a link-symbol in the palette, and then selecting the node symbols in the diagram that are to be linked. In the case of the actor-model, also annotating links with node-symbols should be readily available. Double-clicking on a symbol should rise a description window with detailed data about the symbol.

The menus should at least contain the following choices:

- File
  - New - create a new model.
  - Open - open an existing local model. One can generally chose to open a model for viewing only or for editing.
  - Open old version - enables the traversing of the version hierarchy of the model, and the explicit retaining of a previous version. If this is further worked on, it will be stored as a variant.
  - Save - store a model locally. When storing one might also chose if one wants to publish the model or not.
  - Save as new version - Explicitly save the model as a new version, retaining the old version. One can also here chose to publish the model if anyone currently subscribes to it [4].
  - Close - closes the active model.
  - Check out - check out model from global repository.
  - Check in - check model into global repository.
  - Remove - delete model from repository.
  - Print - print diagram, including setup of printer.
  - Export - export model in a given logical or graphical format.
  - Import - import model from a given export-format.
  - Exit - exit the tool.
- Edit: general editing commands.
  - Undo - support for undoing the n last updates performed on the model.
  - Redo - support for redoing the last undone update.
  - Cut - clears the selected symbols from the model and copies them into a clipboard.
  - Copy - copies the selected symbols into a clipboard.
  - Paste - paste the contents of the clipboard into the current model.
  - Duplicate - duplication of selected symbol.
  - Clear - delete the selected symbol. If the symbol is linked to other symbols, one need to confirm that these links also should be deleted.
  - Replace - replace the selected symbol with the symbol indicated in the palette.
  - Select all - select all the symbols in the model.
- View
  - All - zoom the diagram so that all symbols are visible.

  - Zoom in - show a smaller part of the diagram.
  - Zoom out - show a larger part of the diagram.
  - Normal - Show diagram with normal zoom.
  - Zoom - menu for indicating more specific zoom options, such as zooming relative to the normal zoom, and relative to the current zoom. It should also be possible to indicate that the whole width of the diagram is visible.
  - Inter-model filter : To support the creation of filters. The menus and possibilities here is different for different modeling languages as indicated in Appendix D.3. The general functionality of filters, including the selection of different filters, and the creation of own filters based on the preexisting ones are described in [282] and is not repeated here. If one wants to use the result from a filter to continue modeling, one should explicitly save this as a new version. If not, an attempt to reconciliate the views will be performed when one try to save the model. Although the figures presented this far shows a pure graphical format, it is apparent that one will need a combination of textual and graphical presentation in addition to browsing possibilities of the models that are to model situations on the instance level such as the actor model due to the potentially large set of individuals in an organization. A similar problem with complexity can appear with rule modeling.
  - Intra-model filter: This is applying the use of the links-filters as specified in Appendix D.3, and will take up views to other models according to the link selected. If e.g. a role is selected in the actor-model, one can select to see in a separate window the part of the ONER-model that describe the static aspects of this role. One can also use these views as starting points for further modeling of these models, and the same applies as above for the saving of new versions. The above is similar to the search functionality described by Seltveit in [282].
  - Change internal - update the top internal actor, potentially giving different indications of the type (internal/external) of actors and roles in the model.
- Layout: Functionality to support the layout of the diagram.
  - Snap to grid (on/off) - snap to grid is used for creating better aligned diagrams.
  - Show grid (on/off) - show the existing grid used.
  - Text - moving text within the diagram manually to be able to improve the overall layout.
  - Align - a sub-menu that enables the alignment of selected node-symbols.
  - Spacing - a sub-menu enabling the equal spacing of three or more node-symbols.
  - Aesthetics - a menu to indicate a metric-value for different aesthetics according to the list given in Chapter 6. It should also be able to show the history of such metrics if any.
  - Layout modification - an attempt of improving aesthetics on a chosen metric based on the given constraints. Typical constraints in conceptual modeling will often be that the main nodes, e.g. processes in process models, have similar positions relative to another. These two last points are not developed in PPP as of now, and should be looked upon as ideas for further work.
- Tools: Invocation of more advanced modeling techniques as discussed in Chapter 6.
  - Description - add a description to a selected model element. For an actor, it can be name, indication of if it is internal or external to the indicated top internal actor, and the main role of actor. It should be possible to enter a general narrative, and also the indication of basic information about the source and agreers etc. to this symbol should

be given. The established links to other models should also be listed.
  - Define term - open the project or organization thesaurus to find, enter, or update definitions of terms.
  - Link - based on the selected symbol, explicitly link this to another model. The possible links are different for different symbols.
  - Consistency check - check syntactic quality and logical consistency of the model. For data and process-models this is described in more detail in [196, 340]. For actor-models, rules that can be used for consistency checking are indicated in Chapter 9.1. Overall consistency checking should be performed based on the selection of a set of model, which should be selected on a general level.
  - Completeness assessment - based on the integration between the modeling languages, one should be able to get a completeness assessment based on the driving question technique described in Chapter 9.3.4. It should also be possible to use this technique to ask questions to assess perceived completeness of the model. It should similarly be possible to get such assessments in connection with the remaining work in the creation of a prototype, code-generation, general execution, and the use of the explanation generation facilities based on the current model.
  - Execution - applying either the prototyping functionality, the general execution functionality, and/or the explanation functionality. More detailed tool support for this is indicated in [196], [335], and [114] respectively. When performing prototyping taking many models into account, one must indicate this on the general level. Alternatively, one can specify this by indicating the technical actor to be generated.
  - Code-generation - generation of production code. This will usually be performed on a general level, but also here it is possible to indicate the technical actor that should be created.
  - Model integration - make it possible to select two (or more) models either from an overall project graph, or a version graph, or alternatively chose the current diagram as one of the models, an another model as the other, and support the integration, either by indicating similarities or differences. Additional ideas for how to support this is presented in [4, 282].
  - Report - a sub-menu with a set of different reports that can be created based on the current models. It should be possible to create tailored reports.
- Process: Support the process heuristics:
  - Source - indicate the source of a set of selected statements.
  - Perceived completeness - indicate the perceived completeness, possibly with the help of the completeness assessment techniques.
  - Perceived validity - indicate that an actor perceive a set of statements to be valid. If the actor being chosen is an organizational actor, this is judged as indicating agreement among the social actors being part of the organizational actor.
  - Perceive comprehension - indicate that an actor perceive to understand the meaning of a set of statements.
  - Interest - indicates that an actor other than the source have interest of whole or selected parts of a model.
  - Value - assessment of the value of a selected set of statements.
  - Domain knowledge - indicates that an actor have knowledge of an area that is modeled in a model.
  - Language knowledge - indicates (on the meta-model) the knowledge an actor is re-

garded to have of the modeling language being used.
  - Metrics - A possibility to assess the metrics as described in Chapter 10 on the process heuristics, to support the choice of the action to be done in further modeling. Should be linked up to project management functionality and indication of resource use.
- Option
  - Setup - general setup, such as model type, language for help text and labels.
  - Font - select font for diagram or symbol.
  - Grid size - set grid size.
- Window
  - Cascade
  - Tile
  - Arrange
  - A list of currently available windows
- Help
  - Contents - preferably a hypertext help system. On the help for the modeling constructs, this should be user-specific if possible.
  - Phenomenon - gives explanation of the use of a specific modeling phenomenon that is chosen in a diagram. Should be user-specific if possible.
  - Window - when in a pop-up window, get help with the use of this specific window.
  - Search - search for help on a specific keyword indicating a function etc.
  - Shortcuts - give an overview over speed-keys etc.
  - Using help - help on the use of the help system, conventions etc.
  - Technical support - indications of who to contact in case of technical problems.
  - SIR - form for entering a system investigation report on the CASE-tool, e.g. indicating errors, the need for additional functionality, or general questions.
  - About - general information about the tool.

Appendix E

Statements in PPP

In this appendix, we present a translation of PPP-models into a form of the kind that we outlined in Chapter 6. We have used a meta-model similar to the one used for explanation-generation as presented in Appendix D.2, but have changed and extended this. Since the extensions are not meant to be used for explanation generation, they are not based on RST as the other relationships in the meta-model. Of the same reason, we will only use some of the relationships in Table D.2.

The below is *not* an attempt to present an ultimate representation of a canonical form of models in the PPP-languages, but must rather be looked upon as one of many possible ways of doing this as an outset for further experimentation.

After describing the language, we will present different modeling cases, and how they can be represented in the suggested form.

E.1 Language for the Representation of Statements

We have presented the types of statements below in alphabetic order for easy reference.

- ABTRACTIONLEVEL(PHENOMENA,LEVEL) : Level is here set or element, i.e. indicating if the phenomena that is represented is meant to be an abstraction of an element or a set of elements.
- AGGREGATION(CARTESIAN PRODUCT,SET): Indicate an aggregation.
- ASSOCIATION(SET OF SETS,SET): Set is one of the members of the indicated Set of Sets.
- BEFORE(P1,P2): Phenomena P1 happen before phenomena P2.
- CARDINALITY(SET,CARD): The number of members of a set.
- CLAIM(ILLOCUTIONARY ACT,DOMINANT CLAIM): Dominant claim of a speech act.
- CLASSIFICATION(SET,MEMBER): Indicate the members of a set. This can also be used for indicating explicit domains.
- CONSUME(P1,P2): A phenomena consumes another phenomena.
- DOMAIN(P1,SET): The domain from where values of a phenomena is taken.
- DURATION(E1,TEMPEXP): The temporal duration of a dynamic phenomena.
- DURING(P1,P2): Phenomena P1 happens during the happening of phenomena P2.
- EQUIVALENCE(E1,E2): Two phenomena are regarded to be equivalent.
- EXIST(PHENOMENA,ID) : A phenomena is perceived to exist in some sense by someone. The ID is not necessarily indicated in the model, but are used as an internal identifier
- EXTENSION(P1,P2): Indicates that P2 belongs to P1's extension.
- FROM(L1,P1) : Indicate the start of a directed link

- GENERALIZATION(SETA,SETB): SetA contains all the members of SetB, but not the other way around.
- INFLUENCES(P1,P2): A phenomena influences another phenomena.
- LINK(L1,P1): Indicate an undirected link that links a phenomena with another phenomena.
- MAXCARDINALITY(SET,CARD): The maximum number of members of a set. This can in many cases be indicated using rules.
- MEET(P1,P2): When P1 ends P2 starts immediately.
- MINCARDINALITY(SET,CARD): The minimum number of members of a set, also this can in many cases be indicated using rules.
- MODALITY(PHENOMENA,MODALITY): Modality is one of necessitation, obligation, recommendation, permission, discouragement, prohibition, and exclusion.
- ORDERING(SET,ORDER): Indicates the ordering of an ordered set. Ordering can be temporal or based on values, queues or stacks. When based on values, the ordering can be alphabetic or numeric.
- OVERLAPS(P1,P2): The execution of P1 and P2 overlaps.
- PHENOMENATYPE(ID,PHENOMENATYPE): We have differentiated between the following main types of phenomena: static, dynamic, link, and rule.
- POINT(ILLOCUTIONARY ACT,ILLOCUTIONARY POINT): Illocutionary point of a speech act.
- POSSESSION(E1,E2): E1 can be said to possess E2 from some point of view.
- POSTCONDITION(R2,TFOL): Postcondition of a phenomena written in first order temporal logic.
- PRECONDITION(R1,TFOL): Precondition of a phenomena written in first-order temporal logic.
- PRODUCE(P1,P2): A phenomena produce another phenomena.
- PROPOSITION(ILLOCUTIONARY ACT ,PROPOSITION): Propositional content of a speech act.
- RANGE(L1/R1,SET): The range of values of a link or rule
- REFERENCE(ID,TERM) : A given term is used to reference to the phenomena when communicating about the perception of it. A phenomena can have several such labels.
- SIMEND(P1,P2): P1 and P2 end simultaneously.
- SIMSTART(P1,P2): P1 and P2 start simultaneously.
- SOURCEBEHALF(L1,A1): On behalf of actor A1 of the source of the link L1.
- SOURCEROLE(L1,R1): Role of the source of a link L1.
- TARGETBEHALF(L1,A1): On behalf of actor A1 of the target of the link L1.
- TARGETROLE(L1,R1): Role of the target of the link L1.
- TERMINATE(P1,D1): The appearance of P1 terminate a dynamic phenomena.
- TO(L1,P2): Indicated the end of a directed link.
- TRIGGER(P1,D1): The appearance of P1 activates a dynamic phenomena.

E.2 ONER Models

- A single entity class (E1): EXIST(PHENOMENA,E1), PHENOMENATYPE(E1,STATIC). When using existence, this is to be looked upon as *perceived existence*. All the phenomena are set-phenomena if nothing else is indicated, i.e. an entity class is a static set-phenomena, whereas an entity (object) is a static element.

- Entity class E1 has label 'entity': REFERENCE(E1,ENTITY).
- A single relationship class (R1) : EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,STATIC).
- Relationship-class R1 has label 'relationship': REFERENCE(R1,RELATIONSHIP).
- A single type (T1): EXIST(PHENOMENA,T1), PHENOMENATYPE(T1,STATIC).
- The possible elements of T1 are {E1,E2}: CLASSIFICATION(T1,E1), CLASSIFICATION(T2,E2). One can also specify the domain of the type as domain(T1,D1) where D1 can potentially be infinite.
- Type T1 has label 'type': REFERENCE(T1,TYPE).
- A single method (M1): EXIST(PHENOMENA,M1), PHENOMENATYPE(M1,STATIC).
- Method M1 has label 'method': REFERENCE(M1,METHOD).
- A relationship R1 includes 3 entity classes E1,E2,E3: AGGREGATION(R1,E1), AGGREGATION(R1,E2), AGGREGATION(R1,E3).
- E2 and E3 are subclasses of E1: GENERALIZATION(E1,E2),GENERALIZATION (E1,E3). In addition to this, one have inheritance of relationships, rules, attributes, and methods.
- Subclasses E2 and E3 are distinct: This translates to a rule, which are generally represented in the following statements: exist, phenomenatype, modality, precondition, and postcondition. The pre and postconditions of the rules can be stated freely using the ERL language. In this case:
  EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,RULE) MODALITY(R1,OBLIGATION) (by default a rule is an obligation when nothing else is indicated), PRECONDITION(R1,E2.X AND E3.Y), POSTCONDITION(R1,X≠Y).
  Similar, more complex rules will appear in the case of many classes being a distinct subset.
- Subclasses E2 and E3 make up class E1: EXIST(PHENOMENA,R2), PHENOMENATYPE(R2,STATIC) , MODALITY(R2,OBLIGATION), PRECONDITION(R2,E1.X AND E2.Y AND E3.Z), POSTCONDITION(R2, X=Y OR X=Z).
- Subclasses E2 and E3 is a partition of E1. This is a combination of the above two rules.
- A renamed type: EQUIVALENCE(T1,T2).
- A type T1 is a set of type T2 with indicated cardinality (min:max). The default is (O:N). T1 and T2 are defined as static set-phenomena: ASSOCIATION(T1,T2), MAXCARDINALITY(T2,MAX), MINCARDINALITY(T2,MIN).
- A type T is a composition of N types, for example (T1,T2), and T2 is optional: AGGREGATION(T,T1), AGGREGATION(T,T2), MAXCARDINALITY(T1,1) , MINCARDINALITY(T1,1), MAXCARDINALITY(T2,1), MINCARDINALITY(T2,0).
- A type T1 is a union of T1 and T2: AGGREGATION(T,ONLY-ONE-OF(T1,T2))
- Cardinality-rules of the form E1 np Rel1 1f E2:
  Partial:
  EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,RULE), MODALITY(R1,PERMITTED), PRECONDITION(R1,E1.X AND E2.Y), POSTCONDITION(R1, NOT X REL1 Y).
  Full:
  EXIST(PHENOMENA,R2), PHENOMENATYPE(R2,RULE), MODALITY(R2,OBLIGATORY), PRECONDITION(R2,E1.X AND E2.Y), POSTCONDITION(R2, X REL1 Y).
  1:
  EXIST(PHENOMENA,R3), PHENOMENATYPE(R3,RULE), MODALITY(R3, FORBIDDEN), PRECONDITION(R3, E1.X REL1[E2.Y]), POSTCONDITION(R3, E1.Z REL1[E2.Y] Z≠X).
  N:
  EXIST(PHENOMENA,R4), PHENOMENATYPE(R4,RULE), MODALITY(R4, PERMITTED), PRECONDITION(E1.X REL1[E2.Y]), POSTCONDITION(E1.Z REL1[E2.Y] Z≠X).

- An entity E1 has an attribute A1 with label 'attribute' of type T1: Attributes are represented as a function from E1 to T1, i.e. as a rule: EXIST(PHENOMENA, A1), PHENOMENATYPE(A1,RULE), DOMAIN(A1,E1), RANGE(A1,T1), REFERENCE(A1,ATTRIBUTE). Similarly will it be for attributes for relationships.
- A method M1 is method of E1: POSSESSION(E1,M1).
  Similarly will it be for methods linked to relationships.
- A method has input-types T1 and T2 and output type T3: CONSUME(M1,T), AGGREGATION(T,T1), AGGREGATION(T,T2), PRODUCE(M1,T3).

E.3 PPM

- A single process (P1) : EXIST(PHENOMENA,P1), PHENOMENATYPE(P1,DYNAMIC).
- Process P1 has label 'process': reference(P1, process).
- A single store (S1): EXIST(PHENOMENA,S1), PHENOMENATYPE(S1,STATIC).
- Store S1 has label 'store': REFERENCE(S1, STORE).
- A single timer (T1): EXIST(PHENOMENA,T1), PHENOMENATYPE(T1,DYNAMIC).
- Timer T1 has label 'timer':REFERENCE(T1, TIMER).
- A single actor (A1): EXIST(PHENOMENA,A1), PHENOMENATYPE(A1,DYNAMIC), ABSTRACTIONLEVEL(A1,ELEMENT).
- Actor A1 has label 'actor': REFERENCE(A1, ACTOR).
- A single role (R1): EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,STATIC).
  The role as such is indicated to be passive, since only when it is filled by an actor is it active.
- Role R1 has label 'role': REFERENCE(R1, ROLE).
- A single sink (s1): EXIST(PHENOMENA,S1), PHENOMENATYPE(S1,STATIC).
- A single flow (f1): EXIST(PHENOMENA,F1), PHENOMENATYPE(F1,LINK), ABSTRACTIONLEVEL(F1,SET), ORDERING(F1,TEMPORAL QUEUE).
- The flow f1 has label 'flow': REFERENCE(F1, FLOW).
- Item1 can travel on a flow: EXIST(PHENOMENON,ITEM1), PHENOMENATYPE(ITEM1,STATIC) CLASSIFICATION(F1,ITEM1), ABSTRACTIONLEVEL(ITEM1,ELEMENT).
- Items are described by a (combined) type, which can, but not need to be linked to the ONER-model. I.e. one can say that a flow f1 contains items of type T1, where T1 can be composed and can be described in more detail in the same way as as a type in ONER.
  EXIST(PHENOMENA, L1), PHENOMENATYPE(L1,LINK), LINK(L1,F1), LINK(L1,T1).
- An item on a flow f1 is connected to an illocutionary act IA1. This have an illocutionary point, a propositional content, and a dominant claim. (e.g. request, paper, sincerity), and can in addition be linked to a set of rules. In the example it is only linked to the rule R1: EXIST(PHENOMENA,IA1), POINT(IA1,REQUEST), PROPOSITION(IA1,PAPER), CLAIM(IA1,SINCERITY), EXIST(PHENOMENA,L1), PHENOMENATYPE(L1,LINK), LINK(L1,IA1), LINK(L1,F1), EXIST(PHENOMENA,L2), PHENOMENATYPE(L2,LINK), LINK(L2,IA1), LINK(L2,R1).
  The rules related to the illocutionary acts is expressed through a link between the flow and the rule.
- A conversation C1 consists of four illocutionary acts, IA1, IA2, IA3, and IA4: EXIST(PHENOMENA, C1), PHENOMENATYPE(C1,DYNAMIC), AGGREGATE(C1, IA1), AGGREGATE(C1, IA2), AGGREGATE(C1, IA3), AGGREGATE(C1, IA4).
- The capacity of the channel that is depicted by the flow is n: MAXCARDINALITY(F1,N).
- The flow f1 comes from one phenomena P1 and goes into another P2: FROM(F1,P1),TO(F1,P2).

- A process happens before another process: Since all dynamic phenomena have a potential duration, similar relationships can in principle be indicated between dynamic phenomena as can be indicated between temporal intervals. When a process is linked by a flow that terminates process P1 and triggers process P2, this implies the statement: BEFORE(P1,P2).
- For a process to start, items have to appear on flows in a given pattern which is indicated by the ports and triggering information e.g. PRECONDITION(P1,ITEM1 AND (ITEM2 OR ITEM3)).
- For a process to end, items have to appear on flows in a given pattern which is indicated by the ports and triggering information: POSTCONDITION(P1,ITEM4).
- During processing, the process consumes a set of items according to the port combination: CONSUME(P1,ITEMSET).
- During processing, the process produces a set of items according to the port combination: PRODUCE(P1,ITEMSET)
  Itemset above indicate that it is sets of items in a logical combination that are consumed. These itemsets can be related to items, and one can indicate the cardinality of the itemsets:
  Singular flow: CARDINALITY(ITEMSET,1).
  Repeating flow: MAXCARDINALITY(ITEMSET,N), MINCARDINALITY(ITEMSET,1).
  Conditional flow: MAXCARDINALITY(ITEMSET,1), MINCARDINALITY(ITEMSET,0).
  Both repeating and conditional flow: MAXCARDINALITY(ITEMSET,N), MINCARDINALITY(ITEMSET,0)
- A process P1 is decomposed in processes P1.1 and P1.2: AGGREGATION(P1,P1.1), AGGREGATION(P1,P1.2).
- A process P1 is described by a set of rules R1, R2, being part of a rule-set RS1: POSSESSION(P1,RS1), AGGREGATION(RS1,R1), AGGREGATION(RS1,R2).
- A process P1 is described by a PLD PLD1: POSSESSION(P1,PLD1).
  A PLD can further be described by representing variables as items with a given type, assignments by rules (functions) ordering by BEFORE, iteration by ASSOCIATION, and choice by GENERALIZATION. Receive and send are already represented.
- The behavior of timers can generally be described as rules with no defined modality: EXIST(PHENOMENA,TR1), PHENOMENATYPE(TR1,RULE), PRECONDITION(TR1,PRECONDITION), POSTCONDITION(TR1,POSTCONDITION), POSSESSION(T,TR1).
  In addition do the presence of a timer indicate several other statements:
- A timer has a given delay between sending out items: DURATION(T1,TEMPEXP).
- A clock C1 with no inputs, and one or more outputs: TRIGGER(SYSTEM-START,C1), TRIGGER(OUTFLOWITEM,C1).
- The clock started with the presence of an inflow item: TRIGGER(INFLOWITEM,C1), CONSUME(C1,INFLOWITEM).
- The clock has offflow: TERMINATE(OFFLOWITEM,C1), CONSUME(C1,OFFLOWITEM).
- A delay D1 (including both on/off and outflows): TRIGGER(ONFLOWITEM,D1), CONSUME(D1,ONFLOWIT DURATION(D1,TEMPEXP), TERMINATE(OFFLOWITEM,D1), CONSUME(D1,OFFLOWITEM).
- A flow goes from a store, removing items from the store: CONSUME(F1,STOREITEM).
- A role or actor A1 support a process/store/flow E1: INFLUENCES(A1,E1).
  If the support-link have own properties, it must be represented as is indicated below on the support relationships.

E.4 Actor Models

- A single actor (A1): EXIST(PHENOMENA,A1), PHENOMENATYPE(A1,DYNAMIC), ABSTRACTIONLEVEL(A1,ELEMENT).
- Actor A1 has label 'actor': REFERENCE(A1, ACTOR).
- A single role (R1): EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,STATIC).
  The role as such is indicated to be passive, since it is only when filled by an actor that it is active.
- Role R1 has label 'role': REFERENCE(R1, ROLE).
- A single agent (AG1): EXIST(PHENOMENA,AG), PHENOMENATYPE(A1,DYNAMIC).
- An role or agent A2 is institutionalized by a an actor A1: INFLUENCE(A1,A2).
- A role R1 is a sub-role of role R2: GENERALIZATION(R2,R1).
- An actor A1 fill a role R1: EXTENSION(R1,A1).
  Having hierarchies of roles, this can also indicate several implicit statements.
- An actor A1 is part of another actor A2: AGGREGATION(A2,A1).
- An actor A1 is part of another actor A2 in role R1, or on behalf of A3. The link has description D1. In addition to above, we need to specify the link explicitly: EXIST(PHENOMENA,PL1), PHENOMENATYPE(PL1,LINK), FROM(PL1,A1), TO(PL1,A2), REFERENCE(PL1,D1), SOURCEROLE(PL1,R1), SOURCEBEHALF(PL1,A3).
- An actor A1 supports an actor A2: INFLUENCES(A1,A2).
- An actor A1 supports an actor A2 in role R1 and role R2 or/and on behalf of A11 and A22. The link has description D1: EXIST(PHENOMENA,PL1), PHENOMENATYPE(PL1,LINK), FROM(PL1,A1), TO(PL1.A2), REFERENCE(PL1,D1), SOURCEROLE(PL1,R1), SOURCEBEHALF(PL1,A11), TARGETROLE(PL1,R2), TARGETHALF(PL1,A22).
- An actor A1 communicates with actor A2:
  Similar as for support, but in addition it is possible to also indicate the same things as for flows in PPM. We do not repeat this here.
- An actor or role is internal to an actor: POSSESSION(A1,A2).

E.5 Rules

We have already indicated how to represent simple rules. In addition will we need to look upon the relationship between rules and actors, and rules and rules.

- A rule R1: when τ if ϕ it is ∇ for ρ ψ_1 else ψ_2:
  EXIST(PHENOMENA,R1), PHENOMENATYPE(R1,RULE), MODALITY(R1,∇), INFLUENCES(R1,ρ), PRECONDITION(R1,$\neg \bullet \tau \wedge \tau \wedge \phi$), POSTCONDITION(R1,$\psi_1$):
  EXIST(PHENOMENA,R2), PHENOMENATYPE(R2,RULE), MODALITY(R2,∇), INFLUENCES(R2,ρ), PRECONDITION(R2,$\neg \bullet \tau \wedge \tau \wedge \neg\phi$), POSTCONDITION(R1,$\psi_2$)
- A set of rules RS1 including rules R1 and R2: EXIST(PHENOMENA,RS1), ABSTRACTIONLEVEL(RS1,SET), CLASSIFICATION(RS1,R1), CLASSIFICATION(RS1,R2).
- A rule-relation between two rulesets RS1 and RS2 with modality M: EXIST(PHENOMENA,RR1), PHENOMENATYPE(RR1,LINK), FROM(RR1,RS1),TO(RR1,RS2), MODALITY(RR1,M).
- Argument AR1 is linked to the rule-relationship: EXIST(PHENOMENA,AR1), EXIST(PHENOMENA,AL1), PHENOMENATYPE(AL1,LINK), FROM(AL1,RR1), TO(AL1,AR1).

E.6 Metalevel Statements

- The actor A1 is the source of a statement: SOURCE(A1,STATEMENT).
- The actor A1 agrees to a statement: AGREE(A1,STATEMENT).
- The actor A1 disagrees to a statement: DISAGREE(A1,STATEMENT).

Appendix F

Description of Survey

In the summer and autumn of 1993, a survey-investigation on procedures for development and maintenance of information systems in Norwegian organizations was performed. Two forms, one asking general questions regarding the practices followed in the organization as a whole including 45 questions, and one form asking more detailed questions regarding the maintenance of one existing system including 25 questions were distributed by mail to 350 Norwegian organizations. The organizations were picked from the list of member organizations of DND (Den Norske dataforening) The Norwegian Computer Society. The total number of member organization in DND is approximately 1100, and the forms were sent to the director/manager of the data processing department of the sampled companies. The decision to sample, rather than to survey all of the members, was made on the basis of cost-efficiency. The firms where picked from all over the country, except from the Bergen area. This to avoid asking the same people that had been asked in an earlier survey [130] which contained questions on similar areas.

Accompanying the forms, a following letter asking the respondents to answer the survey signed by the leader of DND at the time was enclosed together with a one page description of the reasons for doing the survey and one page instructions for how to fill out the forms. In addition, all respondents were given a lotto-coupon as a motivator for filling out the forms. The respondents have also received the final report from the survey [171].

Our sample of DND-members implicitly assured that the forms were answered by organizations of some size which look upon CIS support as an important aspect. The respondents were above average regarding size. Whereas the average number of employees of Norwegian organizations within the areas of manufacturing, industry, trade and services are around 8 persons [47], the same average among the respondents of our investigation was around 2350. This means that the results of this survey do not give a picture of the average Norwegian organization, but rather on the situation for computerized information systems support in medium to large organizations according to Norwegian standards.

The first form was supposed to be filled out by the manager himself, whereas it was intended that the second form should be delegated to someone being familiar with the maintenance of the particular system being reported on. Most of the respondents had filled out both forms themselves, but this do not seem to have reduced the quality of the answers.

The contents of the questionary [171] was based on previously performed questionary within this area, especially Lientz/Swanson [193], Swanson/Beath [299], and Henne [130]. The questionary was refined through several pilot fill outs, using both mail and interviews. In addition to this, the questionary went through the academic scrutiny of several of our colleagues before being sent out on a large scale. The pilots were helpful in the process of clarifying the questions being

asked and to limit the size of the investigation.

On some of the questions, we were interested in the quality of the answers, recognizing that some of the figures called for might not be easily obtainable. These had the following qualification possibility:

The above answer is:
Reasonably accurate, based on good data |___|
A rough estimate, based on minimal data |___|
A best guess, not based on any data |___|

Many of the questions in part two were accompanied by the following answer scheme:

Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

In addition it was room for commenting the answer given in prose after most of the questions. This was useful in the coding of the answers to discover potential misunderstandings among the respondents. When misunderstanding were identified, the respondents were contacted for clarification. The forms are presented in full in Section F.5.

F.1 Respondents

Galtung [96] states that the least meaningful survey-size is regarded to be 40 units. Since survey-investigations in the area of development of information systems towards the same population earlier had given a response on 22% [24] and the response to similar surveys in other countries on system maintenance have been around 25% (e.g. [69, 193, 237]), an answer ratio of approximately 20% was expected. This would have given 70 responses. A total of 78 answers were returned, giving an answer ratio of 22%. Some of the answers were negative though, replying that the organization were not doing work of the sort which it was queried about. Other answers also had to be dismissed, giving us a total of 52 valid answers as a basis for analysis. On many of the questions this number is even less though, since not all questions were relevant for all respondents. The number of responses was somewhat less than expected, but was still regarded sufficient for our purpose, which was to get an overall impression on the state of affairs within Norwegian organizations regarding procedures for development and maintenance of CISs.

F.2 Analysis

The results from the survey were after coding entered into an Excel spreadsheet from where it was imported into the statistical package SPSS [236]. SPSS has been used to perform all statistical analysis on the material which are presented in the thesis. The kind of analysis that was performed were hypothesis testing, correlation analysis, and factor analysis in line with what was done in Lientz/Swanson and Nosek/Palvia. When doing hypothesis testing, two main approaches were used:

- Testing for equality of mean of two separate variables.
- Dividing the sample according to the value of some variable, and testing for the equality of means of another variable in each of the sub-samples. Usually some measure of central tendency (mean or median) was used to divide the sample. When significant results appeared from this, a correlation analysis was also performed if possible.

According to the distribution of the variables, statistical significance of the results has been determined either using the twin-tailed Student t-test [236] or the twin-tailed Mann-Whitney test [236]. Statistical techniques are further discussed in Section F.4.

The hypothesis being tested were partly based on earlier investigations, most notably Lientz/Swanson, but also some more exploratory analysis were performed, especially regarding how the amount of functional development and maintenance varied in different samples.

F.3 Critique of the Survey Technique

The survey technique are widely used, but has also been criticized on several accounts by e.g. Jørgensen based on experience with the technique [158, 160].

- Inconsistency in the use of words: In spite of the explicit definition of the term 'software maintenance', the respondents definitions in his investigation were often more exclusive than the questionnaire definition. Because of the different background and working environment of the respondents, this kind of problem is not surprising having social construction theory in mind. This effect was also experienced in our investigation when it came to what was included in the term application system maintenance. To address this the questions for assessing the amount of resources used on maintenance and development also including a split-up on these numbers to indicate what work-areas that were assessed to be included into the maintenance figures. The numbers for functional maintenance would neither have been different if the respondents had specified functionally perfective maintenance as development, which was what most of the people with a different view did. This problem is not addressed to the same extent on many of the other questions. Some precautions performed here was the use of pilot-studies and the use of control questions to test that the understanding coincided.
- Inconsistency in opinion: Asking maintainers and managers the same questions showed that they often differed significantly in opinion [158]: Several papers studying human judgement have reported a tendency to underestimate the impact of the environment and to overestimate the impact of one's own role. Seen from the overview of our respondents, we have also recorded the managerial view. On most of the questions that we have reported on here (such as number of employees, number of systems being developed in-house etc) this is not regarded as problematic. As illustrated by Jørgensen, this problem appeared on the breakdown of maintenance figures, where corrective maintenance where over-assessed by some managers. The same effect has been reported by Arnold [10]. Similar effects can have been present on other questions, especially those asking for subjective judgments. This effect could have been discovered by using a conscious post-investigation evaluation, but this was unfortunately not done, and is a weakness in our research methodology.
- Incompleteness and incorrectness: Some of the answers were partly incomplete also in our study. Some incorrect answers were also investigated. For important data, the respondent was contacted on telephone to support the additional data, but this was only done in a few cases. For important data, the respondents were asked to give an assessment of the quality of this data, which was used by us to see if this led to biased data.

F.4 Statistical Methods

The following statistical methods were used to analyze the data from the survey investigation:

- Hypothesis testing
- Correlations

The descriptions of these techniques are taken from [24, 144, 234, 236].

F.4.1 Hypothesis Testing

The purpose of hypothesis testing is to help draw conclusions about population parameters based on results observed in a random sample. The procedure remains virtually the same for tests of most hypotheses.

- A hypothesis of no difference (of mean value) called the null hypothesis (H_0) and its alternative are formulated.
- A test statistic is chosen to evaluate the null hypothesis. The test statistics used to test hypothesis where we compare the results from our investigation with the results from other investigations was the two-sample Student-T test. This is calculated as follows:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}}}$$

  where $\overline{X_1}$ is the sample mean of group 1, S_1^2 is the variance and N_1 is the sample size. Based on the sampling distribution of the above statistics, one can calculate the probability that a difference at least as large as the one observed would occur if the two population means are equal. This probability is called the observed significance level. If the observed significance level is small enough (usually less than 0.05 or 0.01) the hypothesis that the population means are equal is rejected.
  The test statistics used when comparing sub-populations in our investigation is the paired-samples T test. Although the interpretation of the significance of results from paired experiments is the same as those from the two independent samples discussed previously, the actual computation is different. For each pair of cases, the difference in the responses is calculated. The statistics used to test the hypothesis that the mean difference in the population is 0 is

$$t = \frac{\overline{D}}{S_D/\sqrt{N}}$$

  where $\overline{D}$ is the observed difference between the two means and S_D is the standard deviation of the differences of the paired observations.
- For the sample, the test statistics is calculated
- The probability, if the null hypothesis is true, of obtaining a test value at least as extreme as the one observed is determined.
- If the observed significance level is judged small enough, the null hypothesis is rejected. When validating the null-hypotheses, it is possible to do two types of error:

  1. Rejecting H_0 when it is correct
  2. Accepting H_0 when it is incorrect

  The significance level (denoted as p) expresses the probability of doing error 1.

It is most appropriate to use the t-test when the dependant variable is close to normally distributed, and when the variance of the two subgroups are close to each other. The sample size should be above 30. Two measures of the normality of a distribution is skewness and kurtosis.

Skewness is defined by

$$\frac{\sum((X_i - \overline{x})/s)^3}{n}$$

This express the degree of symmetry of the distribution. A symmetric distribution has a skewness of 0. A positive value tells that the distribution is skewed to the left, whereas a negative value tells that it is skewed to the right.

Kurtosis is defined by

$$\frac{\sum((X_i - \overline{x})/s)^4}{n} - 3$$

A normal distribution have a kurtosis of 0. A positive value means that the curve is pointier, whereas a negative value means that it is flatter than a normal distribution.

Two additional tests for normality being used is the Lilliefors test and the Shapiro-Wilks's test.

When it appears that the data is not normally distributed, it is instead possible to use non-parametric tests such as the Mann-Whitney test (also known as the Wilcoxon test). The data can be measured either on an interval or ordinal scale. The test requires only that the observations be a random sample and that values can be ordered from smallest to largest. The hypothesis tested is similar to that of the two independent samples t-test. However, normality and equality-of-variance assumptions are not needed.

F.4.2 The Correlation Coefficient

The correlation coefficient is often useful to quantify the strength of the association between two variables. One commonly used measure is the **Pearson's correlation coefficient**, denoted by r. It is defined as

$$r = \frac{\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y})}{(N-1)S_x S_y}$$

where N is the number of cases and S_x and S_y are the standard deviations of the two variables. The absolute value of r indicates the strength of the linear relationship with 1 as the largest possible value. This occurs when all points fall exactly on the same line. When the line has a positive slope, the value of r is positive. When the slope of the line is negative, the value of r is negative. A value of 0 indicates no relationship. Note that the values can have a strong association, but a small correlation coefficient if the relationship is not linear.

The Pearson's correlation is appropriate only for data that attain at least an interval level of measurements and the variables should have normal distribution. This indicates that an nominal and ordinal scale is not applicable for such analysis. On the other hand, several researchers have stated that ordinal scales can be used in correlation analysis. Boyle [33] states that using an ordinal scale in parametric statistics will be conservative regarding the real correlation coefficient used with an interval-scale. The requirement for a normal distribution can also be relaxed a bit according to Guildford [112], but the distribution should be unimodal (i.e. have one maximum) and have symmetric properties.

The number of cases in the investigation should be above 30 [24]. Significance level in the correlation can be explained as following. The probability of that there do not exist a linear correlation relationship between the variables is equal or less than respectively 0.01 and 0.001, where 0.01 and 0.001 are the significance levels. This significance level can also be explained from a null hypotheses H_0:It exists no simultaneous variation among the variables.

For large samples sizes, even very small correlation coefficients have small observed significance levels. When evaluating the relationships among variables, both the value of the coefficient and its associative significance level should be investigated. Test of significance is most useful for small samples (less than 100). Having large samples will make rather weak correlations and associations significant, and it is then more interesting to concentrate on the correlation-value [144]. When performing correlation analysis, Bergersen [24] demands the value of r to be at least 0.3 with a significance level of 0.01, to consider the relationship important. Van Swede and van Vliet [315] on the other hand uses a r-level of 0.25 with significance levels of 0.05 and 0.01. This demand is followed in our investigation. Lientz and Swanson uses $r \geq 0.1$ with significance levels of 0.01 and 0.001 in their investigation which is somewhat surprising taking into account their sample size and the discussion above.

For ordinal data or interval data that do not satisfy the normality assumption, another measure of the linear relationship between two variables, Spearmans' rank correlation coefficient, is available. The rank correlation coefficient is the Pearson's correlation coefficient based on the ranks of the data if there are no ties (adjustments are made if some of the data are tied). If the original data for each variable have no ties, the data for each variable are first ranked, and then the Pearson's correlation coefficient between the ranks of the variables is computed. The interpretation is the same as for Pearson's correlation coefficient, except that the relationship between ranks and not values, is examined.

A common mistake in interpreting correlation coefficients is to assume that correlation assume causation. No such conclusion is automatic. Similarly, the absence of significant correlation does not have to mean that correlations do not exist.

F.5 Questionnaire

The following appendix contains the English version of the questionnaire. The translation of the questions from the original Norwegian forms is biased towards the formulation of the questions in the Lientz/Swanson and Swanson/Beath investigations as appropriate, i.e. when the question in the Norwegian forms was in fact a translation of the question in the American investigations in the first place.

Part I

Name : _____

Occupation : |___| Manager |___| Project leader
: |___| Systems developer

Employment status : |___| Permanent |___| Temporary
: |___| Consultant

Education : _____
Years of computer experience : _____

Short description of experience, tasks and responsibilities in the last five years : _____

1. **Which of the following activities do you regard as being part of maintenance of information systems? (Check one or more as appropriate)**

   a. Correcting errors in systems in production. |___|
   b. Adapting the system to a changed technical architecture. |___|
   c. Developing new functionality in existing systems. |___|
   d. Improving non-functional properties (e.g. performance). |___|
   e. Developing new systems with functionality similar to the one found in old systems. |___|
   f. Developing new systems in new functional areas. |___|

2. **In your opinion, how demanding is the problems which arise in application systems maintenance, when compared to those which arise in new application systems development?**

   a. New system development by far the more demanding |___|
   b. New systems development somewhat more demanding |___|
   c. Maintenance and new systems development equally demanding |___|
   d. Maintenance problems somewhat more demanding |___|
   e. Maintenance problems by far the more demanding |___|

Organization : _____
Department : _____

3. **Type of organization:**

a.	Banking	____
b.	Building	____
c.	Real estate	____
d.	Energy	____
e.	Finance	____
f.	Publishing	____
g.	Packing	____
h.	Insurance	____
i.	Wholesale	____
j.	Trade	____
k.	Interest organization	____
l.	Mechanical industry	____
m.	Food production industry	____
n.	Government	____
o.	Petroleum industry	____
p.	Production industry	____
q.	Process industry	____
r.	Shipping	____
s.	Transportation	____
t.	Media	____
u.	Computer industry	____
v.	Other	____ Please indicate: _____

4. **How many employees has the organization?** _____

5. **What is the annual budget of the IS-organization in 1993 including hardware, software and personnel (in million Norwegian kroner)**

a.	More than 5:	____
b.	between 4 and 5:	____
c.	between 3 and 4:	____
d.	between 2 and 3:	____
e.	between 1 and 2:	____
f.	between 0.5 and 1:	____
g.	less than 0.5 :	____

6. **In terms of the total person-hours worked annually, how much is used for application systems maintenance?**
   |____| %

   How much of this is used for:

   a. Correcting errors in systems in production. |____|
   b. Adapting the system to a changed technical architecture. |____|
   c. Developing new functionality in existing systems. |____|
   d. Improving non-functional properties (e.g. performance). |____|

   The above answer is:
   Reasonably accurate, based on good data |____|
   A rough estimate, based on minimal data |____|
   A best guess, not based on any data |____|

7. **As above, how much is used for application systems development**
   |____| %

   How much of this is used for:

   a. Developing new systems with functionality similar to the one found in old systems. |____|
   b. Developing new systems in new functional areas. |____|

   The above answer is:
   Reasonably accurate, based on good data |____|
   A rough estimate, based on minimal data |____|
   A best guess, not based on any data |____|

8. **What is the total number of equivalent full-time personnel in the IS-organization?**
   |____| persons

9. **How many of these work as systems developers?** |____| persons

10. **What is the length of service distribution of the current application staff in the IS-organization?**

    0-1 years |____| persons
    1-3 years |____| persons
    3-6 years |____| persons
    6-10 years |____| persons
    More than 10 years |____| persons

11. **What is the distribution of educational backgrounds for the applications staff?**

Graduate college degree, specialization in computer science	|____| persons
Graduate college degree, other specialization	|____| persons
Bachelors college degree, specialization in computer science	|____| persons
Bachelor college degree, other specialization	|____| persons
No higher education	|____| persons

12. **What is the distribution of immediate prior job experience of the current application staff?**

Position in other I.S. organization within parent organization	|____| persons
Other position within parent organization	|____| persons
Position in other I.S. organization, not in parent organization	|____| persons
Other position, not in parent organization	|____| persons
No prior position (student)	|____| persons

13. **How many (full-time equivalent) systems development consultants are used in a year?**
\|____\| persons

14. **What is the distribution of own system developers and consultants on application development and maintenance projects**

Only own people on project	|____| %
One or two consultants, rest own people	|____| %
More consultants, but own people in majority	|____| %
Equally many consultants as own people on project	|____| %
Consultants in majority	|____| %
Almost only consultants	|____| %

<u>The above answer is:</u>

Reasonably accurate, based on good data	|____|
A rough estimate, based on minimal data	|____|
A best guess, not based on any data	|____|

15. **Are those applications programmers and/or systems analysts responsible for the maintenance of existing application systems organized separately from those responsible for new application system development?**

    |___| Yes |___| No

16. **Is maintenance of the application system performed by those who made the original system?**

    Always __ 5 __ 4 __ 3 __ 2 __ 1 Never __ Do not know

    Comment : _____
    _____
    _____

17. **Is maintenance of application systems performed as a training activity in the organization?**

    Always __ 5 __ 4 __ 3 __ 2 __ 1 Never __ Do not know

    Comment : _____
    _____
    _____

18. **What is the total number of major systems in the current installed application systems portfolio?** |___|

    Comment : _____
    _____
    _____

19. **What is the application domain of the major systems of the current installed application systems portfolio ?**

    _____
    _____
    _____
    _____

20. **What is the size of the user population served by the major systems in the current application system portfolio?** |___| persons

21. **What is the age distribution (measured from date of original installation) of the current installed application system portfolio?**

0-1 years	|____| systems
1-3 years	|____| systems
3-6 years	|____| systems
6-10 years	|____| systems
More than 10 years	|____| systems

22. **What is the distribution of development backgrounds of the current installed application systems portfolio?**

Developed by the IS organization	|____| systems
Developed by user organization	|____| systems
Developed by outside firm	|____| systems
Package with large internal adjustments	|____| systems
Package with small internal adjustments	|____| systems

Comment : _____
_____
_____

23. **Of the major systems in the current installed application system portfolio, how many rely on other systems for their input data?**
\|____\| systems

Comment : _____
_____
_____

24. **How many systems relies upon major systems to obtain their data?** \|____\| systems

Comment : _____
_____
_____

25. **What hardware and system software is used to support the application system portfolio? (List the configurations and the number of major application systems supported by each)**

Hardware/system software configurations	Number of systems
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

26. **What programming languages are used within the current installed application system portfolio?**

Language	Number of systems
COBOL	_____
Assembler	_____
PL/1	_____
FORTRAN	_____
PASCAL	_____
C	_____
4GL	(indicate) _____
Other	(indicate) _____

No.systems

27. **What is the total number of major approved new systems currently under development, for future installation in the application system portfolio ?**
|____| systems

28. **Of the total number of major approved new systems currently under development, how many of these are replacement systems (for systems currently in the application system portfolio?)**
|____| systems

29. **What is the age distribution of the systems to be replaced ?**

1-3 years	|____| systems
3-6 years	|____| systems
6-10 years	|____| systems
More than 10 years	|____| systems

30. **In the case of systems to be replaced in the current installed application system portfolio, what are the important reasons for replacement?**

	5	4	3	2	1
a. Excessive burden to maintain					
b. Excessive burden to operate					
c. Excessive burden to use					
d. Existence of application package alternative					
e. Existence of application generator alternative					
f. Changes in hardware and system software					
g. Standardization with rest of organization					
h Integration with other new or existing systems					
i. Other (Please indicate)					

Marking:
5: Extreme importance
4: Substantial importance
3: Moderate importance
2: Slight importance
1: No importance

31. **When developing replacement systems and new systems with overlapping functionality with existing systems, to what extent is it possible to reuse code?**

32. **When developing replacement systems and new systems with overlapping functionality with existing systems, to what extent is it possible to reuse specifications and design?**

33. **What organizational controls are established for the maintenance of the application systems?**

a. All user requests for changes to the application system must be logged and documented |___|
b. Change requests are classified according to type and importance |___|
c. All user requests for changes to the application system must be cost justified |___|
d. All changes to the application program must be logged and documented |___|
e. All changes to the application programs must undergo a formal retest procedure |___|
f. With the exception of emergency fixes, all changes to the application programs are batched for periodic implementation |___|
g. At acceptance test of changes, checks are also performed to ensure that the documentation is also updated |___|
h. User requesting changes are always kept informed of the status of their change request |___|
i. The same routines are used for changes request coming from IS-organization and from the user organization |___|
j. A formal audit of the application system is made periodically |___|
k. Equipment costs associated with operating and maintaining the application system are charged back to the user |___|
l. Personnel costs associated with operating and maintaining the application system are charged back to the user |___|

Comment : _____
_____
_____

34. **Do the IS-organization use a complete development methodology covering all phases of application systems development?**
|___| Yes |___| No

35. **Is the methodology externally developed or is it developed internally by the IS-organization?**

|___| External |___| Internal

Comment : _____
_____
_____

36. **If the method is internally developed, what is the foundation for it?**

_____
_____
_____
_____

CASE (Computer Aided Software Engineering) as used below contain all forms of automated tools for development and maintenance of application systems.

37. **Is CASE-technology used in the development of new application systems?**
|___| Yes |___| No
If so - what product(s) is used _____

38. **Is CASE-technology used in the maintenance of application systems?**
|___| Yes |___| No
If so - what product(s) is used _____

39. **For how long has these tools been used in the organization?**
|___| year

Comment : _____
_____
_____

40. **How many of the major systems in the current installed application systems portfolio are supported by the CASE-tools?**
|___| systems

Comment : _____
_____
_____

41. **Which of the following functions are currently supported by the use of CASE-tools?**

a.	Conceptual modeling (ER, DFD etc.)	⌊___⌋
b.	Drawing of screens and reports	⌊___⌋
c.	Prototyping/simulation for validation	⌊___⌋
d.	Consistency checking of specifications	⌊___⌋
e.	Storing, administration and reporting of system information	⌊___⌋
f.	Code generation	⌊___⌋
g.	Generation of DB-schema	⌊___⌋
h.	System test	⌊___⌋
i.	Reverse engineering	⌊___⌋
j.	Project and process management	⌊___⌋

Comment : _____

_____

_____

42. **If code generation is supported, how much of the systems is on the average generated automatically, and what parts are generated?**

⌊___⌋ % _____ parts

The above answer is:

Reasonably accurate, based on good data ⌊___⌋

A rough estimate, based on minimal data ⌊___⌋

A best guess, not based on any data ⌊___⌋

43. **For the parts generated by the code generator, is maintenance performed on the specifications for later regeneration, or are they performed directly on the code?**

⌊___⌋ On specifications ⌊___⌋ On code

Comment : _____

_____

_____

44. **The use of formalized methods and CASE-tools is meant to result in increased productivity and quality in the development and maintenance of application systems. In your opinion, to what degree is the following important in this respect?**

		5	4	3	2	1
a.	Increase the productivity of the developer					
b.	Support rapid prototyping and validation					
c.	Simplify the development process					
d.	Formalize/standardize the development process					
e.	Reduce the time for application development					
f.	Reduce the cost of application development					
g.	Improve the interface toward the system					
h.	Integration of development phases and tools					
i.	Automatic generation of documentation					
j.	Standardization of documentation					
k.	Increased possibility of reuse					
l.	Improved maintainability					
m.	Automatic code generation					
n.	Better control of the application development					
o.	Automating the project management					
p.	Automatic consistency checking					
q.	Reduce the number of errors in the system					
r.	Fulfill user requirements					
	Other (Please indicate)					

Marking:
5: Very important
3: important
1: Not important

Comment : _____
_____
_____

45. **Overall, in your judgment, to what extent are the following a problem in maintaining the current installed application system portfolio?**

		5	4	3	2	1
a.	Turnover of maintenance personnel					
b.	Quality of documentation					
c.	Changes in hardware and system software					
d.	User demands for enhancements					
e.	Skills of maintenance personnel					
f.	Quality of original application system					
g.	Availability of maintenance personnel					
h.	Competing demands for maintenance personnel					
i.	Lack of user interest in application system					
j.	Application system run failures					
k.	Lack of user understanding of system					
l.	Storage requirements					
m.	Processing time requirements					
n.	Motivation of maintenance personnel					
o.	Maintenance programming productivity					
p.	Hardware and system software reliability					
q.	Data integrity in application system					
r.	Unrealistic user expectations					
s.	Lack of adherence to programming standards					
t.	Budgetary pressures					
u.	Inadequate training of user personnel					
v.	Turnover in user organization					
w.	Lacking management support of application system					
	Other (Please indicate)					

Marking:

5: Major problem
4: Somewhat major problem
3: Minor problem
2: Somewhat minor problem
1: No problem at all

Comment : _____
_____
_____

Part II

Name : _____

Occupation : |___| Manager |___| Project leader
: |___| Systems developer

Employment status : |___| Permanent |___| Temporarily
: |___| Consultant

Education : _____
Years of computer experience : _____

Short description of experience, tasks and responsibilities in the last five years : _____

For this part of the investigation should a major application system which is important for the organization and that has been in use for more than a year be used as an outset

Name of application system: _____

Reason for choice:

_____
_____
_____
_____
_____

1. **Is an overview of who is asking for changes in a system being kept?**

   Always __ 5 __ 4 __ 3 __ 2 __ 1 Never __ Do not know

   Comment : _____
   _____
   _____

2. **Is statistics being made over this?**

   Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

   Comment : _____
   _____
   _____

3. **Are the users informed on the status of change requests?**

   Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

   Comment : _____
   _____
   _____

4. **Are the users informed about the scope and impact of changes before they are implemented?**

   Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

   Comment : _____
   _____
   _____

5. **Is statistics made for how long time change request are put on hold?**

   Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

   Comment : _____
   _____
   _____

6. **Is it possible to trace the costs of changes to the system back to the users?**

   Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

   Comment : _____
   _____
   _____

7. **Are the consequences of changes properly assessed?**

   Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

   Comment : _____
   _____
   _____

8. **Is time and cost assessed for individual changes?**

   Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

   Comment : _____
   _____
   _____

9. **Are maintenance tasks categorized by type?**

   Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

   Comment : _____
   _____
   _____

10. **Are maintenance tasks categorized by importance?**

    Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

    Comment : _____
    _____
    _____

11. **Do changes proposed by people in the IS-organization undergo the same kind of control as those proposed by the user community?**

    Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

    Comment : _____
    _____
    _____

12. **Is statistical information about errors and change requests collected and stored for each maintenance task?**

    Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

    Comment : _____
    _____
    _____

13. **When changes are implemented, are also parts of the systems which are not obviously influenced by the change re-tested?**

    Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

    Comment : _____
    _____
    _____

14. **Are modules that are not authorized ever put in production?**

    Often ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

    Comment : _____
    _____
    _____

15. **Are naming standards used in the development and maintenance of specifications, design and programs?**

    Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

    Comment : _____
    _____
    _____

16. **Is the adherence to these standards controlled?**

    Always ___ **5** ___ **4** ___ **3** ___ **2** ___ **1** **Never** ___ **Do not know**

    Comment : _____
    _____
    _____

17. **Does it exist detailed guidelines that outlines when a program should be rewritten instead of being further maintained?**

To a large degree ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 No guidelines ___ Do not know

Comment : _____
_____
_____

18. **When changing a program, is also other documentation, including specification and design updated?**

Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

Comment : _____
_____
_____

19. **Is it possible to identify different versions of different modules, and legal configurations of these versions?**

Always ___ 5 ___ 4 ___ 3 ___ 2 ___ 1 Never ___ Do not know

Comment : _____
_____
_____

20. **What is the total number of programs currently included in the application system maintained?**

_____ programs

The above answer is:
Reasonably accurate, based on good data |___|
A rough estimate, based on minimal data |___|
A best guess, not based on any data |___|

21. **What is the total number of source language statements currently included in the application system maintained?**

_____ lines

The above answer is:
Reasonably accurate, based on good data |___|
A rough estimate, based on minimal data |___|
A best guess, not based on any data |___|

22. **How much of this is comments?**

Less than 1%	|___|
1% - 5%	|___|
5% - 10%	|___|
10% - 20%	|___|
More than 20 %	|___|

The above answer is:

Reasonably accurate, based on good data	|___|
A rough estimate, based on minimal data	|___|
A best guess, not based on any data	|___|

23. **Of the total number of source language statements currently maintained, what percentage is written in each of the following languages?**

Language	Percentage
COBOL	|___| %
Assembler	|___| %
PL/1	|___| %
FORTRAN	|___| %
PASCAL	|___| %
C	|___| %
4-GL	(indicate) _____
Other	(indicate) _____

The above answer is:

Reasonably accurate, based on good data	|___|
A rough estimate, based on minimal data	|___|
A best guess, not based on any data	|___|

24. **How many pre-defined user reports are currently associated with the application system maintained?**

\|___\| reports

The above answer is:

Reasonably accurate, based on good data	|___|
A rough estimate, based on minimal data	|___|
A best guess, not based on any data	|___|

25. **Of the total number of person-hours now expended annually on maintenance of the application system, what percentage is expended in each of the following problem areas?**

a.	Emergency program fixes	____ %
b.	Routine debugging	____ %
c.	Accommodations of changes to hardware and system software	____ %
d.	Enhancements for users	____ %
e.	Changed functionality	____ %
f.	Improvement of program documentation	____ %
g.	Recoding for efficiency	____ %
h.	Cosmetic changes of user interface and reports	____ %
i.	Preventive maintenance	____ %
h.	Other (Please indicate) _____	____ %

The above answer is:

Reasonably accurate, based on good data	____
A rough estimate, based on minimal data	____
A best guess, not based on any data	____

Appendix G

Mathematical Symbols

In this appendix, we list the mathematical notation used in the thesis, in particular in connection with the quality-framework and the rule-language.

Symbol	Meaning
$\mathcal{S}$	Set
$2^{\mathcal{S}}$	Powerset
$\#\mathcal{S}$	Cardinality i.e. number of members of a set
$\emptyset$	A set with no members
$\subset$	Proper subset of set
$\subseteq$	Subset of set
$\not\subset$	Not subset of set
$\in$	Element of set
$\notin$	Not element of set
$\equiv$	Equivalent to
$\not\equiv$	Not equivalent to
$\setminus$	Complement set
$\cup$	Set union
$\cap$	Set intersection
$\neg$	Negation
$\wedge$	Logical and
$\vee$	Logical or
$\rightarrow$	Implication
$\diamondsuit$	Sometime in past
$\diamondsuit$	Sometime in future
$\blacksquare$	Always in past
$\square$	Always in future
$\bullet$	Just before
$\circ$	Just after
$\mathcal{U}$	Until
$\mathcal{S}$	Since
τ	Trigger

ϕ	Condition
ϕ_s	State condition
ψ	Consequence
ψ_a	Action
ψ_s	State
ρ	Role
α	Actor
$\mathcal{E}$	$\neg \bullet \tau \wedge \tau \wedge \phi$
∇	Deontic operator
O	Obligatory
R	Recommended
P	Permitted
D	Discouraged
F	Forbidden
∇_ρ (ψ / $\neg \bullet \tau \wedge \tau \wedge \phi$)	General rule
N	Necessary
E	Excludes
t_R	Real time
t_M	Temporal module time
$\mathcal{A}$	Audience, the technical and social actors that must relate to a model
A_i	A member of the audience
$\mathcal{D}$	The set of all statements that can be stated about a problem at hand
$\mathcal{I}$	The set of all statements which the audience think that a model consist of
$\mathcal{K}$	The set of statements regarded relevant among the participants.
$\mathcal{K}_i$	All possible statements that would be correct and relevant for addressing the problem at hand according to the explicit knowledge of the participant A_i
$\mathcal{K}^i$	The statements of the explicit internal reality of the social actor A_i
$\mathcal{L}$	The statements that can be expressed in a given (set of) languages
L_i	A language
$\mathcal{L}_F$	Statements expressible in a set of formal (operational) languages
$\mathcal{L}_I$	Statements expressible in a set of informal languages
$\mathcal{L}_S$	Statements expressible in a set of formal (logical) languages
$\mathcal{L}_S$	Statements expressible in a set of semi-formal languages
$\mathcal{L}_i$	The statements that can be stated in language L_i
$\mathcal{M}$	The set of statements in an externalized model
$\mathcal{M}_E$	The set of explicit statements in an externalized model
$\mathcal{M}_I$	The set of implicit statements in an externalized model
$\mathcal{M}_i$	A model based on the knowledge of social actor A_i
$\mathcal{M}_{L_i}$.	A model written in language L_i
$\mathcal{M}^i$	The statements in a model which are relevant for audience member A_i
$\mathcal{M}(L_i)$	The language model of language L_i
$\mathcal{M}(\mathcal{D})$	The model of a given domain
$\mathcal{P}$	The individual social actors of the audience
$\mathcal{S}$	When used about actors, the stakeholders to the modeling, else a general set
$\mathcal{V}$	A viewspec of a model

Appendix H

Terminology

We give in this appendix a comprehensive overview our the terminology used in this thesis. There is some overlap between this appendix and Chapter 2.1. We have also included some of the abbreviations being used in the thesis, without further explanation.

Terms are grouped in the following areas:

- Time.
- Phenomena.
- State and rules.
- Data, information, and knowledge.
- Language and models.
- Actors and activities.
- Systems.
- Social construction.
- Methodology.

In some cases the definition of a term is found after that it has been used in another definition. An alphabetical overview of the terms are given separately at the end of the appendix.

H.1 Time

The definitions in this area are to a large extent pragmatic, to be able to use them in a well-defined manner in the later definitions. Thus, we are not entering into philosophical and quantum mechanical aspects of the nature of time.

Time points: Time can be represented by time points, such that the only relation between time points other than identity is that one time point precedes the other.

Time interval: A time interval is the ordered pair of *time points* (the begin and end-point of the interval) such that the first either precedes or is equal to the other.

Time scale: A time scale divides *time* into coherent *time intervals.*

Time unit: The smallest *time interval* that can be represented on a given *time scale* is termed the time unit.

Duration: The duration of a *time interval* is the number of consecutive *time units* between the one after the one in which the begin point of the *time interval* occurs, until and including the *time unit* in which the end point of the *time interval* occurs. A *time point* has no duration.

Figure H.1 illustrates the terms discussed in this section.

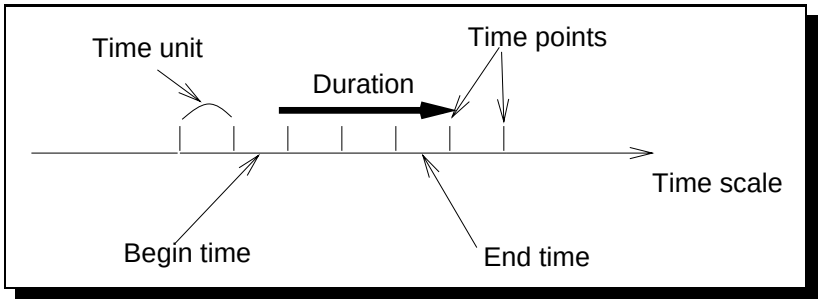

Figure H.1: Time and duration

H.2 Phenomena

A phenomenon is used as the elementary unit of the terminology. In other similar terminologies, the term 'thing' is used in this respect [195, 323].

Phenomenon: A phenomenon is something as it appears in the mind of a person. The world is perceived by persons to consist of phenomena. A phenomenon can be perceived to exist independently of the perceiving person, or be perceived to be a purely mental.

With the phrase 'something which is perceived to exist independent of the perceiver', we mean something that the person in question regards as existing external to him, for instance another person. An idea of a new CIS that only appears in someone's mind though have no such "real-world" equivalence, at least not until it is externalized.

Relevance: A *phenomenon* is of relevance to a non-empty set of persons in a *time interval* if it is of interest to all members of the set in the *time interval.*

Potential relevance: A *phenomenon* is of potential relevance to a non-empty set of persons in a *time interval* if it is of interest to at least one of the persons in the set in the *time interval.*

Relevance is socially and temporally constrained which are as expected taking social construction into account. Relevance needs the notion of shared explicit knowledge (see below) to be meaningful i.e. if there are no phenomena which are perceived equally by two persons, no phenomena will be relevant.

Domain: A domain is defined as the source of any kind of mapping.

'Domain' includes the meaning known from algebra, but the term 'mapping' is used in a slightly more general sense than usual. Not only sets can be mapped into sets as in mathematics, but also areas into areas. When used in the mathematical sense, a domain will be a finite or infinite set of values.

Property: A property is an aspect of a *phenomenon* which can be described and given a value. A *phenomenon* will have a set of *potentially relevant properties.* The values for the properties are members of the *domains* for these properties. All *phenomenon* have at least one property, namely its perceived individual existence or lack thereof.

Type: A non-empty set of *properties* which together characterize certain *phenomena.*

Subtype: A subtype S of a *type* T is a set of *properties* such that T is a proper subset of S.

Supertype: A subset of a type

Class: The set of all *phenomena* of a certain *type.* These *phenomena* is called the **members** of the class.

Subclass: The subclass S of a class T is the proper subset of the class T such that the phenomenon in S has a type which are a subtype of the type of the phenomena of T.

When having several sub-classes of a class you can have different cases based on coverage and disjointness.

A set of *subclasses* of a *class* **cover** the *class* if all *members* of the *class* are *members* of at least one of the *subclasses.*

A set of *subclasses* of a *class* are **disjoint** if no *members* of a *subclass* are *members* of any of the other *subclasses* of the *class.*

A set of *subclasses* which are both *disjoint* and *cover* the *class* is called a **partition** of the *class.*

Environment: The environment of a *phenomena* is the set of *actors* which *acts upon* it.

H.3 State and Rules

State: The state of a *phenomenon* is the set of mappings of all *properties* of the *phenomenon* into values from the *domain* of the *properties.* A *phenomenon* can only be in one state within a *time unit.*

State space: The state space of a *phenomenon* is the set of all possible *states* of the *phenomenon.* All *subsystems* of a *system* have its own state space.

Transition: A transition is a mapping from a domain comprising *states* to a co-domain comprising *states.*

Event: An event is a change of *state* of a *phenomena.* It is effected through a *transition.* An event happen within a *time unit,* i.e. it has a zero *duration.*

Trigger: A trigger is a relationship between an *event* and one or more *activities* and expresses the perceived cause for an *actor* to carry out the *activities.*

History: The history of a *phenomenon* is the chronologically ordered *states* of the *phenomenon.*

Rule: A rule is something which influences the *actions* of a non-empty set of *actors.* A rule is either a rule of necessity or a deontic rule [327].

The term rule is used to cover more situations than what is usually found, since it also includes what is often referred to as guidelines or instrumental rules [250].

Rule of necessity: A rule of necessity is a *rule* that must always be satisfied. It is either analytic or empirical (see below).

Analytic rule: A *rule of necessity* which can not be broken by an inter-subjectively agreed definition of the terms used in the rule is called analytic.

Example: 'The age of a person is never below 0'

Empirical rule: A *rule of necessity* that can not be broken according to present *shared explicit knowledge* is called empirical.

Although not as strongly necessary as an analytic rule, this kind of rules are rules that can be treated as if they are rules of necessity, and one would not expect them to be broken.

Example: 'Nothing can travel faster than the speed of light'

Deontic rule: A *rule* which is only socially agreed among a set of persons. A deontic rule can thus be violated without redefining the terms in the rule. A deontic rule can be classified as being an obligation, a recommendation, a permission, a discouragement, or a prohibition [180].

'$\delta\epsilon o\upsilon\tau\omega\varsigma$' is Greek and means " as it should be". The inclusion of recommended and discouraged above is novel compared to traditional deontic logic [321], but has been included in newer frameworks for deontic logic e.g.[155, 156].

Constitutive rule: A *deontic rule* which applies to *phenomena* that exist only because the rule exist [278].

Generally, this kind of rule can be written: A counts as B in context C. When using a general rule-format, the context is included in the precondition.

Static rule: A *rule* restricting the allowable *states* of a *phenomenon* is called static.

Dynamic rule: A *rule* restricting the allowable *state transitions* of a *phenomenon* is called dynamic.

Both rules of necessity and deontic rules can be classified as being static or dynamic.

Lawful transition: A *transition* is lawful if it obey the *dynamic rules of necessity* regarding the *phenomenon.*

Deontic transition: A transition is deontic if it is *lawful* and also obey the *dynamic deontic rules* regarding the *phenomenon.*

Lawful state space: The set of *states* of a *phenomenon* that comply with the *static rules of necessity* concerning the *phenomenon* is termed the lawful state space.

Deontic state space: The set of *states* of a *phenomenon* that are *lawful* and in addition comply with the *static deontic rules* concerning the *phenomenon* is termed the deontic state space.

Internal event: An *event* that arises in a *phenomenon* by virtue of a *lawful or deontic transition* in the *phenomenon* is called internal.

External event: An external event is an *event* that arises in a *phenomenon* by virtue of the *act* of an *actor* in the *environment* of the *phenomenon.*

Stable state: A *state* in which a *phenomenon* will remain unless forced to change by virtue of an *external event.*

Unstable state: A *state* that will change into another *state* by virtue of an *internal event* is called unstable.

H.4 Data, Information, and Knowledge

Knowledge: Knowledge is defined as that which is accepted as valid, relevant, and true by a person.

Knowledge is by definition linked to the individual person. It can be divided into *explicit* and *tacit knowledge.*

Explicit knowledge: Explicit knowledge is the awareness of a person of *properties* and values of *properties* of *phenomena.*

This indicates that it is both explicit knowledge to be aware of a persons height in cm, and that 'height' is a *relevant property* of a person. Since also a person is a *phenomenon,* the explicit knowledge he or she has can be looked upon as part of the *state* of this *phenomenon* since a *potential relevant property* of a person can be that he is able to know something, and what he or she knows. Through infinite introspection this could indicate that the knowledge of a person is infinite, but since it is seldom relevant with this kind of introspection, this is not found problematic. Explicit knowledge can be more or less *precise, certain,* and *complete.*

Example on precision: That someone knows that a city has 2,433,775 inhabitants at a certain time is more precise explicit knowledge than if the same person know that a city has around 2.5 million inhabitants.

Example on completeness: That someone knows that Oslo is the capital of Norway is less complete knowledge of Oslo than to also know that Oslo is located in the southern part of Norway.

Example on certainty: A poor farmer in Kuala Lipis who once have heard about the city of Bombay and that it lies in India has less certain explicit knowledge than someone who have been there himself. If you hear something several times from several different people, your certainty of some explicit knowledge will usually increase. If you have only read it once in a tabloid newspaper, your certainty of the "fact" is usually lower.

In FRISCO [195], explicit knowledge is termed information. We define the term 'information' differently below.

Shared explicit knowledge: Shared explicit knowledge is an inter-subjectively agreed identical awareness of some *properties* and values of *properties* of *phenomena* by two or more persons which have been achieved through a process of social construction.

Tacit knowledge: Tacit knowledge is *knowledge* that can not be represented externally to the person and only shows up in the *actions* of the person having the knowledge.

It is possible to differentiate between two kinds of tacit knowledge. That which could have been represented externally, but which one either choose not to, or can not find the appropriate *symbols* for, and so-called true tacit knowledge.

Information: Information is externalized *explicit knowledge* which are not already known by the person who receives it, i.e. a *state transition* for a person appears when he receives information, thus receiving information can be looked upon as an *event.*

This means that information is socially and temporally constrained. If I already know something (and know that I know it) , I do not perceive to receive information if I am told the same thing again (even if the certainty of the knowledge might increase). Thus our definition is hopefully close to the one used in everyday language as illustrated in [195] "Information is what you get or may get if you ask certain kinds of questions.... Answers to such questions are often provided at some information desk."

Symbol: The *explicit knowledge* of a person can be externalized in a persistent form using symbols.

Message: A set of related *symbols* expressed in a *language* transmitted by an *actor* intended for a non-empty set of *actands.* The set of *actands* which ultimately receives the message can be empty.

Communication: The exchange of *messages* between *actors.*

Sign: A sign is the triplet (*symbol*, person, *phenomenon*), i.e. a sign is *symbol* that represent a *phenomenon* for a person.

Data: Data are *symbols* that can be preserved, transformed, and transported by a computer. Data and other *symbols* can be internalized as *knowledge* by persons.

H.5 Language and Models

Language: A set of *symbols*, the **graphemes** of the language being the smallest units in the writing system capable of causing a contrast in meaning, a set of **words** being a set of related *symbols* constituting the **vocabulary** of the language, rules to form **sentences** being a set of related words (**syntax**), and some inter-subjectively agreed definitions of what the different sentences mean (**semantics**).

In a natural language e.g. English, the symbols and words will be ordered linearly, whereas in a diagrammatic language symbols are ordered spatially. In addition to the aspects described above, one also often talks about the **pragmatics** when discussing languages, being the relationship between symbols, words, and sentences and the effect these have on persons.

Statement: A *sentence* representing a single *property* of a certain *phenomena.*

Language extension: The set of all *statements* that can be made according to the *graphemes, vocabulary*, and *syntax* of a *language.*

Natural language: A natural language is the *language* of a cultural society (for instance a tribe or a nation) - It is usually learned and applied from childhood by the set of persons belonging to the society.

Professional language: A professional language is a *language* used by a set of persons working in a certain kind of area or in a scientific discipline. Usually such a language is not learned before the person has been active in the area for a while.

Formalism: A formalism is a formal *language*, i.e. a *language* with a precisely defined *vocabulary*, *syntax*, and *semantics.*

The semantics can be **operational** and/or **logical**. If the semantics is based on mathematical logic, we use the term **logical formalism**. If it is possible to execute a set of sentences in the language on a computer, the language is said to have an operational semantics.

All formalisms are professional languages.

Semi-formal language: A semi-formal language is a *language* with a precisely defined *vocabulary* and *syntax*, but without a precisely defined *semantics.*

Also semi-formal languages (e.g. DFD) are professional languages.

Informal language: An informal language is neither *formal* nor *semi-formal. Natural languages* are of this category, and also a *professional language* can be informal.

Abstraction: An abstraction is the *phenomenon* of a set of *phenomena* and its *properties* at some level of approximation. The abstraction contains incomplete *explicit knowledge* about the *phenomena*, i.e. there are more to know about the *phenomena* than is in the abstraction. This do not mean that the abstraction can not contain all *relevant knowledge* in a given *time interval.*

Classification: The *abstraction* where individual *phenomena* are grouped together in a *class* based on perceived common *properties.*

Example: “Rod Steward” and “Mick Jagger” can be grouped together in the class “singers”.

Aggregation: An *abstraction* which is a Cartesian product of *classes.*

Example: A bicycle being built up from wheels, a seat, a frame, handlebars etc.

Generalization: An *abstraction* which is a subset of the union of a set of *classes.*
Example: Both employees and customers are persons.

Association: An *abstraction* which is a set of *classes.*
Example: The classes “Men” and “Woman” are members of the set “sex-groups”.

Model: A model is an *abstraction* externalized in a *language.*
A model is assumed to be simpler than, resemble, have the same structure and way of functioning as the phenomena it represent.

Conceptual model: A *model* of a *domain* made in a formal or semi-formal language with a limited vocabulary.
Comment: Many conceptual modeling languages are partly diagrammatic, in which case they are logographic, but this is not looked upon as a requirement.

Language model: The *model* of a *language.*
Within conceptual modeling, this is often termed ’meta-model’, which is only a correct term when looking upon it from the point of view of repository-management for a modeling-tool.

System model: A *model* of a *system.*

H.6 Actors and Activities

A *phenomenon* is **acted** upon by another *phenomenon* if its *history* is different from what it would have been if the other *phenomenon* did not influence it.

Actor: An actor is a *phenomenon* that *acts upon* another *phenomenon,* the **actand** .

Acquaintances: The acquaintances of an *actor* is the set of *actors* that either *acts upon* or is *acted upon* by the *actor.*

Social actor: A social actor is an *actor* that includes at least one person. Social actors might be **individual** or **organizational** (see below).

Technical actor: A technical actor is an *actor* that do not include any persons.
Technical actors can be **computational** and **temporal** . Other subtypes of actors might for instance be production actors, but these will not be discussed here.
Whereas temporal actors are some time-measuring device (i.e. a clock of some sort), computational actors are either **hardware** actors or **software** actors. *Computational actors* are either **atomic** or **systemic** including *atomic* and *systemic subsystems.* Computational actors can be said to be compatible in the following meanings:

- **Hardware compatibility:** Stating which *hardware actors* that can *act* upon each other.
- **Executional compatibility:** Describe which *software actors* that can be executed on which *hardware actors.*

- **Software compatibility:** Stating which *software actors* that can *act* upon each other.

Software actors can be versions of 0:N other software actors, i.e. a software actor can be recreated by performing a set of state changes to the actor it is a version of. A set of state changes in this meaning is called a **delta**. The original actor is called a **predecessor** of the version actor, whereas this is called a **successor** of the original actor. Software actors might have several predecessors and successors. These relations are transitive. Two or more software actors that have the same immediate predecessor are termed **variants** .

Software actors are **supportive** or **applicative** relative to an *organization*. The difference is that applicative actors are being customized to some degree to cater for the specific needs of the organization. For instance will a customized order-entry application system be regarded as applicative, whereas an underlying commercial database system which is used by the order-entry system is supportive. Subtypes of supportive actors are [254]:

- **General supportive software actors** are *software actors* that potentially support all *social actors* in an *organization*. Examples of classes of these are operating systems, file-handlers, and window-systems.
- **Office supportive software actors** are *software actors* supporting office work, example of classes of these are word processors, database systems, graphics tools, spreadsheets, communication programs, and statistics programs.
- **Developer supportive software actors** are *software actors* that typically only support developers of *applicative software actors* directly. Examples of classes of such systems are compliers, CASE-tools, debuggers, general modeling tools, building tools, versioning tools, and test tools.

Internal actor: *Actors* being **internal** to an *organization* are *actors* being part of the *organizational system* of the *organization* in one or more of the relevant *roles* they are currently filling.

External actor: *Actors* being **external** to an *organization* are *actors* not being part of the *organizational system* of the *organization* in any of the relevant *roles* they are currently filling.

Individual social actor: A person interacting with his *environment* is termed an individual social actor.

Organizational social actor: An organizational actor is a *social actor* which consists of a set of more than one person performing goal-oriented and co-ordinated action. An organizational actor can also include *technical actors*, but this is not mandatory.

Permanent organizational actor: An *organizational actor* for which a *begin time-point* of its existence can be perceived, but normally not the future *end time-point*.

Temporary organizational actor: An *organizational actor* for which both the *begin time-point* and the possibly future *end time-point* of its existence can be perceived.

Periodic organizational actor: An *organizational actor* for which a set of *begin time-points* and (possibly future) *end time-points* of its existence can be perceived, and where there is normally the same *time-interval* between the different *begin time-points*. The *duration* of this *time-interval* is longer than the individual lifetime of the *organizational actor*.

Reincarnation: The creation of a *periodic organizational actor.*

Action: An action is the *phenomenon* of one *phenomenon acting* upon other *phenomena.*

Activity: An activity is a *system* of *actions.*

Stakeholder: The stakeholders of an *activity* are the set of persons who perceive or is perceived by other persons to potentially lose or gain from the *activity.*

Participant: The participants of an *activity* are the set of persons who act upon the *actands* of the *activity* as part of the *activity.*

Process: A process is an *activity* which takes a set of *phenomena* and transforms them into a possibly empty set of *phenomena.*

Behavior: Behavior is defined as a time series of *activities.*

Role: *Behavior* that can be expected by an *actor* by other *actors.*

Agent: An *actor* acting in a particular *role.*

Formal role: A *role* where part of the expected *behavior* of an *actor* filling the *role* is institutionalized by an *organizational actor.* A typical example of a formal role is a position such as a professor. All roles have usually also two additional aspects:

- The informal part of the *role.* Expectations to an *actor* filling the *role* which are not institutionalized. e.g. a professor is absent-minded.
- The expectation to an *agent,* because of the particular *actor* filling the *role.*

Role conflict: Inconsistent expectations to an *actor* because of filling two or more *roles* or because of differing expectations to a role that the actor fill from two or more other actors.

H.7 Systems

System: A system is a set of correlated *phenomena,* which itself is a *phenomenon.* Each *phenomenon* that is contained in the system is said to be **part** of the system. A system has at least one systemic *property* not possessed by any of its parts.

The following example taken from [195] indicates the necessity of the requirement of a systemic property: If you buy some eggs from a farmer and use two of them for breakfast, then the correlated phenomena: You, the farmer, the farmers hen that laid the eggs, the frying pan you used to prepare the eggs, and the two eggs now in your stomach could fit as a system using a definition not including a systemic property.

System viewer: A person who perceives the *system* as a *phenomenon.*

Subsystem: A subsystem of a *system* is a *system* that is part of another *system*, the set of *phenomena* being part of the subsystem is a proper subset of the set of *phenomena* being part of the whole *system*.

Subsystem structure: A *partition* of a *system* into a set of *subsystems* together with a set of correlations among the *subsystems*.

Constructivity: The *relevant properties* of a *system* can be derived from the *relevant properties* of the *subsystems* of the *system* given the *subsystem structure* of the *system* [293].

Constructivity should not be confused with social construction theory.

Active system: A *system* where at least one of the *subsystems* is an *activity* is called an active system.

Passive system: If there is no *subsystem* in the *system* that is an activity, it is called a passive system.

Open system: A *system* is open if it has an *environment*.

Information system (IS): An information system is a *system* for the dissemination of *data* between persons, i.e. to potentially increase their *knowledge*.

Data system: A data system is a *system* to preserve, transform, and transport *data*.

A data system is usually a sub-system of an *information system*. Both data systems and *information systems* may be contained in the *domain* they convey *data* about.

Organization: An organization is defined as a non-empty set of persons, and other *phenomena* which is a *phenomenon* where goal-oriented and co-ordinated *action* is aimed at.

An organization is an *organizational actor* when interacting with other *phenomena*.

Organizational system: An organizational system is a *system* having the *actors* and *activities* of an *organization* as *subsystems*.

Organizational information system (OIS): An OIS is the *information system* for the dissemination of *data* within an *organization*. The OIS is a *subsystem* of an *organizational system*.

Computerized information system(CIS): A CIS is an *information system* which are based on the use of computers for the dissemination of *data*.

User (of a CIS): A user of a *CIS* is someone who potentially increases his or her *knowledge* about some *phenomena* other than the *CIS* with the support of the *CIS*. An **end-user** increases his and hers *knowledge* in areas which are *relevant* to him independently of the actual *CIS* by *interacting* with the *CIS*. **Indirect users** of a *CIS* increase their *knowledge* by getting results from the *CIS* without directly *interacting* with the actual *CIS*.

Computerized organizational information system (COIS): A COIS is a *system* for the dissemination of *data* within an *organization* which are based on the use of computers. This is a *subsystem* of the *OIS* of the organization.

The COIS contains the set of *internal software actors* which support the *internal social actors* of the *organization*, and the *hardware actors* these *software actors* are executed on.

Application system: An application system is a *subsystem* of the *COIS* being adapted to the needs of the *organization.*

When an application system interact with its *environment* it is an *applicative actor.*

(Application system) portfolio: The portfolio of an *organization* is the set of *application systems* in the *COIS* of the *organization.*

Dynamic system: A *system* that always is in a *state* from which there exist a *lawful transition.*

Static system: A *system* that is not *dynamic* is called *static.*

H.8 Social Construction

Definitions of terms from social construction theory as they are used in the thesis are given here. The definitions are based on [102].

Local reality: The *local reality* of a person is the way the person perceives the world that he or she *acts* in.

In addition to the persons *knowledge* this also includes feelings and values of the person.

Externalization: The enactment of *local reality.* The most important ways *social actors* externalize their *local reality*, are to speak and to construct *languages*, artifacts such as *models*, and institutions such as *rules.*

Organizational reality: That which guides and controls persons *actions* in an *organization.*

Internalization: Making sense out of the *actions*, institutions, artifacts etc. in the *organization*, and making this *organizational reality* part of the individual *local reality* of a person.

Organizational closure: A *process* of social construction where the *actors* keep reproducing the same *organizational reality.*

H.9 Methodology

The terms underneath are defined here in the context of conceptual modeling for CIS-support in organizations.

Conceptual modeling: The *activity* of constructing *conceptual models.*

Audience: The *actors* that need to relate to the *conceptual models* constructed during *conceptual modeling*

Method: A method is a set of *rules* for creating *models* with a *language.*

Approach: An approach consists of a non-empty set of semi-formal or formal *languages* and a number of *rules* for using these *languages* to construct *models.*

(Model) verification: The *process* of assuring whether a *model,* created according to a certain *approach,* conforms to the *rules of necessity* of the *language* used, or has the expected *semantic.*

(Model) validation: The *process* of assuring that a *model* corresponds to the *explicit knowledge* of those *social actors* which are the source of the *model* .

Whereas verification is potentially decidable, validation is not so, one can never be 100% certain that the externalization in the form of a conceptual model correspond to the local reality of an individual. Even though, validation is a useful activity, due to the possibility for falsification, i.e. one can say that a model do not correspond to one's internal reality.

(Model) transformation: A *process* where a *model* written in a *language* is transformed into another *model* in the same *language.*

Statement insertion: A *transformation* where the resulting *model* contains *statements* that are not contained in the original *model.*

Statement deletion: A *transformation* where the resulting *model* do not contain *statements* that are contained in the original *model.*

Syntactically valid statement deletion: A *statement deletion* resulting in a *model* being conformant to the *syntax* of the *language* the *model* is written in.

(Model) layout modification: *Transforming* a *model* into another *model* containing the same *statements.*

(Model) filtering: *Transforming* a *model* into another *model* containing a subset of the *statements* of the original model.

A model filtering consist of a set of *statement deletions.*

Syntactically valid (model) filtering: *Model filtering* resulting in a *model* being conformant to the *syntax* of the *language* of the model.

(Model) translation: A *process* where a *model* written in a *language* is transformed into another *model* written in a (set of) different *language(s).*

Rephrasing: A *transformation* where some of the implicit *statements* of a model are made explicit.

Paraphrasing: A *translation* where the involved *languages* are textual.

Visualization: A *translation* where the source *language* is textual, and the target *language* is diagrammatic.

Code-generation: A computer-supported *translation* where the target *language* have an *executional semantics.*

Complete translation: A *translation* where all *statements* in the source *model* is also contained in the target *model.*

Valid translation: A *translation* in which all *statements* in the target *model* is also contained in the source *model.*

Prototype: An executable *model* of (parts of) an *information system* which emphasizes specific aspects of that *system.*

Development of an application system in an organization: The *process* of producing a new *application system* in the *organization* based on the current *OIS* and the *knowledge* of *internal* and potentially *external actors.*

Development is divided into two categories.

- Development of **replacement systems** being *application systems* that replace existing *application systems*, and offer the same functionality as the already existing *application systems.*
- Development of *application systems* covering functional areas that are not currently supported by the existing *COIS.*

Maintenance of an application system in an organization: The *process* of creating an updated *version* of an *application system* used in the *organization* through a temporally ordered set of *lawful transitions* based on an existing *application system* and the knowledge of internal and potentially external *actors.*

Corrective maintenance: *Maintenance* performed to identify and correct processing failures, performance failures and implementation failures in an *application system.*

Adaptive maintenance: *Maintenance* performed to adapt *application systems* to the changes among the *supporting technical actors* of the *application system.*

Perfective maintenance: *Maintenance* performed to enhance performance, change or add new functionality, or improve future maintainability of the *application system.* Perfective maintenance is divided into **functional and non-functional perfective maintenance** based on the effects of the performed changes. Functional changes are changes to the functions offered by the *application system* or said differently, how *end-users* can potentially increase their *knowledge* using the *application system.* Non-functional changes implies changes where the quality features

of the *application system* and other features being important for the developer and maintainer of the *application system* such as modifiability are improved.

Functional development: *Development* or *maintenance* where changes in the *application system* increases the functional coverage of the *portfolio* of the *organization.* This includes *development* of new *application systems* which covers areas which are not covered by the existing *COIS*, and also includes *functional perfective maintenance.*

Functional maintenance: Work made to sustain the functional coverage of the *portfolio* of the *organization.* This includes the three other types of *maintenance*, but also includes the *development* of *replacement systems.*

Devtenance in an organization: The *process* of producing an updated *version* of the *COIS* through a temporally ordered set of *lawful transitions* based on the existing *OIS* and the *knowledge* of *internal* and potentially *external actors.*

Methodology: A *system* of *rules, approaches,* and *computational actors* to aid *development and/or maintenance* of *application systems.*

H.10 Abbreviations

This section contains a set of abbreviations used in the thesis. These are listed here in full, although we are not giving any further explanation of the term.

4GL: 4. generation language.

ABC: Actor, bank, channel (in ABC-models).

ABC: Alle bidrar til consensus (in ABC-method).

AD: Actor dependency.

ALBERT: Agent-oriented language for building and eliciting real-time requirements.

AM: Actor model.

ARIES: Acquisition of requirements and incremental evolution of specifications.

BNM: Behavioral network model.

BPR: Business process re-engineering.

CASE: Computer aided software engineering.

CATWOE: Customer, actor, transformation, weltanschauung, owner, environment.

CFP: Call for papers.

CIM: Computer integrated manufacturing.

COISIR: *COIS* investigation report.

COM: Component object model.

CONFORM: Configuration management formalization for maintenance.

CORBA: Common object request broker architecture.

CR: Change request.

CRC: Camera ready copy.

CSCW: Computer supported cooperative work.

DAIDA: Development of advanced interactive data intensive applications.

DBMS: Data base management system.

DFD: Data flow diagram.

DND: Den norske dataforening.

DoD: (The American) department of defense.

DRL: Deontic rule language.

DSM: Domain structure model.

ECML: Executable conceptual modeling language.

EDFD: Entity data flow diagrams.

EIS: Existing information system.

ER: Entity relationship.

ERAE: Entity, relationship, attribute, event.

ERL: External rule language.

ERT: Entity relationship time.

ESPRIT: European strategic programme for research and development in information technology.

ETHICS: Efficient technical and human implementation of computer-based systems.

F3: From fuzzy to formal.

FCIS: Future CIS.

FG: Functional grammar.

FRISCO: Framework for information systems concepts.

GSM: Generic semantic model.

HOQ: House of quality.

IBIS: Issue based information systems.

ICASE: Integrated *CASE.*

IDE: Interactive Development Environments.

IDT: Institutt for datateknikk og telematikk.

IFIP: International federation of information processing.

ISO: International standards organization.

JAD: Joint application design.

LLD: Language for legal discourse.

NATURE: Novel approaches to theories underlying requirements engineering.

NFR: Non-functional requirement.

NTH: Norges teknisk høyskole.

OLE: Object linking and embedding.

OMT: Object modeling technique.

ONER: Our new *ER* modeling language.

OO: Object-oriented.

OOA: Object-oriented analysis.

OOD: Object-oriented design.

OORASS: Object-oriented role analysis, synthesis, and structuring.

PD: Participatory design.

PID: Process interaction diagram.

PLD: Process life description.

PPM: Process port modeling.

PPP: Phenomena, process, program.

RDD: Responsibility driven design.

REBOOT: Reuse based on object-oriented techniques.

RIN: Role interaction network.

RST: Rhetorical structure theory.

SAMPO: Speech act based office modeling approach.

SA/RT: Structured analysis, real time.

SCM: Software configuration management.

SEI: Software engineering institute.

SD: Standard deviation.

SDL: Semantic data language.

SMM: Software maintenance model.

SPSS: Statistical package for social science.

SQL: Structured query language.

SSADM: Structured systems analysis and design method.

STD: State transition diagrams.

STP: Software through pictures.

STEPS: Software technology for evolutionary participative systems development.

SSM: Soft systems methodology.

TEQUEL: Temporal query language.

UDD: User interface dialog description.

UID: User interface description.

UIP: User interface presentation.

WWW: World wide web.

Index

4GL485

ABC485
abstraction477
acquaintances478
act478
actand478
action480
active system481
activity480
actor478
  applicative479
  atomic478
  computational478
  external479
  hardware478
  individual478
  internal479
  organizational478
    periodic479
    permanent479
    temporary479
  social478
    individual479
    organizational479
  software478
  supportive479
  systemic478
  technical478
  temporal478
AD485
adaptive maintenance484
agent480
aggregation477
ALBERT485
AM485
analytic rule474
application system482
  development484
  maintenance484
applicative actor479
approach483
ARIES485
association478
atomic computational actor478
audience483

behavior480
BNM485
BPR485

CASE485
CATWOE486
CFP486
CIM486
CIS481
class473
classification477
closure482
code-generation484
COIS482
COISIR486
COM486
communication476
compatibility
  executional478
  hardware478
  software479
complete translation484
computational actor478
computerized information system481
  organizational482
computerized organizational information system482
conceptual model478
conceptual modeling482
CONFORM486
constitutive rule474
constructivity481
CORBA486

corrective maintenance484
CR486
CRC486
CSCW486

DAIDA486
data476
data system481
DBMS486
delta479
deontic rule474
deontic state space475
deontic transition474
development484
    functional485
devtenance485
DFD486
DND486
DoD486
domain472
DRL486
DSM486
duration472
dynamic rule474
dynamic system482

ECML486
EDFD486
EIS486
empirical rule474
end-user481
environment473
ER486
ERAE486
ERL487
ERT487
ESPRIT487
ETHICS487
event473
    external475
    internal475
executional compatibility478
explicit knowledge475
external actor479
external event475
externalization482

F3487
FG487
filtering483
FIS487
formal role480
formalism477
    logical477
FRISCO487
functional development485
functional maintenance485
functional perfective maintenance484

generalization478
grapheme476
GSM487

hardware actor478
hardware compatibility478
history474
HOQ487

IBIS487
ICASE487
IDE487
IDT487
IFIP487
indirect user481
individual social actor478, 479
informal language477
information476
information system481
    computerized481
    organizational481
internal actor479
internal event475
internalization482
interval471
IS481
ISO487

JAD487

knowledge475
    explicit475
    shared explicit476
    tacit476

language476
    informal477
    natural477

professional477
semi-formal477
language extension477
language model478
lawful state space475
lawful transition474
layout modification483
LLD487
local reality482
logical formalism477

maintenance484
adaptive484
corrective484
functional485
perfective484
functional484
non-functional484
member473
message476
method483
methodology485
model478
conceptual478
filtering483
language478
layout modification483
system478
transformation483
translation483
validation483
verification483

natural language477
NATURE487
NFR487
non-functional perfective maintenance484
NTH487

OIS481
OLE488
OMT488
ONER488
OO488
OOA488
OOD488
OORASS488
open system481
organization481
organizational closure482
organizational information system481
organizational reality482
organizational social actor478, 479
organizational system481

paraphrasing484
part480
participant480
partition473
passive system481
PD488
perfective maintenance484
periodic organizational actor479
permanent organizational actor479
phenomenon472
PID488
PLD488
portfolio482
potential relevance472
PPM488
PPP488
pragmatics477
predecessor479
process480
professional language477
property473
prototype484

RDD488
REBOOT488
reincarnation480
relevance472
potential472
rephrasing483
replacement system484
RIN488
role480
formal480
role conflict480
RST488
rule474
analytic474
constitutive474
deontic474
dynamic474
empirical474

necessity474
static474
rule of necessity474

SA/RT488
SAMPO488
SCM488
SD488
SDL488
SEI488
semantics476
logical477
operational477
semi-formal language477
sentence476
shared explicit knowledge476
sign476
SMM489
social actor478
software actor478
software compatibility479
SPSS489
SQL489
SSADM489
SSM489
stable state475
stakeholder480
state473
stable475
unstable475
state space473
deontic475
lawful475
statement477
deletion483
insertion483
static rule474
static system482
STD489
STEPS489
STP489
subclass473
cover473
disjoint473
partition473
subsystem481
subsystem structure481
subtype473
successor479
supertype473
supportive actor479
symbol476
syntax476
system480
active481
application482
data481
dynamic482
information481
open481
organizational481
passive481
replacement484
static482
system model478
system viewer480
systemic computational actor478

tacit knowledge476
technical actor478
temporal actor478
temporary organizational actor479
TEQUEL489
time interval471
time point471
time scale471
time unit471
transformation483
transition473
deontic474
lawful474
translation483
complete484
valid484
trigger473
type473

UDD489
UID489
UIP489
unstable state475
user481
end481
indirect481

valid translation484
validation483
variants479
verification483
visualization484
vocabulary476

word476
WWW489

Bibliography

[1] W. W. Agresti. What are the new paradigms? In *New Paradigms for Software Development*, pages 6–10. IEEE, 1986.

[2] V. Alagappan and W. Kozaczynski. The evolution of very large systems. In Lowry and McCartney [204], pages 1–24.

[3] L. E. Allen and C. S. Saxon. A-Hohfeld: A language for robust structural representation of knowledge in the legal domain to build interpretation-assistance expert systems. In Meyer and Wieringa [223], pages 52–71.

[4] R. Andersen. *A Configuration Management Approach for Supporting Cooperative Information System Development*. PhD thesis, IDT, NTH, Trondheim, Norway, 1994.

[5] R. Andersen, J. A. Bubenko jr., and A. Sølvberg, editors. *Proceedings of the Third International Conference on Advanced Information Systems Engineering (CAiSE'91)*, Trondheim, Norway, May 1991. Springer-Verlag.

[6] R. Andersen, R. Conradi, J. Krogstie, G. Sindre, and A. Sølvberg. Project courses at the NTH : 20 years of experience. In J. L. Diaz-Herrera, editor, *7th Conference on Software Engineering Education (CSEE'7)*, pages 177–188, San Antonio, USA, 1994. Springer Verlag (LNCS 750).

[7] Andersen Consulting. *FOUNDATION - Method/1, Tools Reference Manual, Version 2.1*, 1989.

[8] Andersen Consulting. *Method/1, System Development Management*, 1995.

[9] A. L. Antón, W. M. McCracken, and C. Potts. Goal decomposition and scenario analysis in business process reengineering. In Wijers et al. [330], pages 94–104.

[10] R. S. Arnold and D. A. Parker. The dimensions of healthy maintenance. In *Proceedings of the 6th International Conference on Software Engineering (ICSE)*, pages 10–17. IEEE Computer Society Press, September 13–16 1982.

[11] P. Atzeni and R. Torlone. A metamodel approach for the management of multiple models and the translation of schemas. *Information Systems*, 18(6):349–362, 1993.

[12] J. H. August. *Joint Application Design: The Group Session Approach to System Design*. Yourdon Press, 1991.

[13] E. Auramäki, R. Hirschheim, and K. Lyytinen. Modelling offices through discourse analysis: A comparison and evaluation of SAMPO with OSSAD and ICN. *The Computer Journal*, 35(4):342–352, 1992.

[14] E. Auramäki, R. Hirschheim, and K. Lyytinen. Modelling offices through discourse analysis: The SAMPO approach. *The Computer Journal*, 35(4):342–352, 1992.

[15] E. Auramäki, E. Lehtinen, and K. Lyytinen. A speech-act-based office modelling approach. *ACM Transactions on Office Information Systems*, 6(2):126–152, April 1988.

[16] J. L. Austin. *How to do things with words*. Harvard University Press, Cambrige MA, 1962.

[17] D. E. Avison and A. T. Wood-Harper. *Multiview: An Exploration in Information Systems Development*. Blackwell, Oxford, England, 1990.

[18] S. C. Bailin. An object-oriented requirements specification method. *Communications of the ACM*, 32(5):608–623, May 1989.

[19] Å. Bakkane. Code generation in the PPP I-CASE environment. Master's thesis, IDT, NTH, Trondheim, Norway, February 1991.

[20] R. Balzer et al. Operational specifications as the basis for specification validation. In Ferrari et al., editor, *Theory and Practice of Software Technology*. 1983.

[21] V. R. Basili. Viewing maintenance as reuse-oriented software development. *IEEE Software*, 7(1):19–25, January 1990.

[22] K. M. Benner. The ARIES simulation component (ASC). In *Proceedings of the Eight Knowledge-Based Software Engineering Conference (KBSE'93)*, September 1993.

[23] P. Berger and T. Luckmann. *The Social Construction of Reality: A Treatise in the Sociology of Knowledge*. Penguin, 1966.

[24] L. Bergersen. *Prosjektadministrasjon i systemutvikling. Aktiviteter i planlegningsfasen som påvirker suksess (In Norwegian)*. PhD thesis, ORAL, NTH, Trondheim, Norway, 1990.

[25] P. Bieber and F. Cuppens. Computer security policies and deontic logic. In Meyer and Wieringa [223], pages 52–71.

[26] B. I. Blum. A taxonomy of software development methods. *Communications of the ACM*, 37(11):82–94, November 1994.

[27] B. Boehm, P. Bose, E. Horowitz, and M. J. Lee. Software requirements negotiation and renegotiation aids: A theory-W based spiral model. In *17th International Conference on Software Engineering (ICSE'95)*, pages 243–253, Seattle, Washington, USA, April 23–30 1995.

[28] B. W. Boehm. *Software Engineering Economics*. Prentic Hall, 1981.

[29] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72, May 1988.

[30] C. Boldyref, E. L. Burd, and R. M. Hather. An evaluation of the state of the art for application management. In Müller and Georges [228], pages 161–169.

[31] A. Borgida. Features of languages for the development of information systems at the conceptual level. *IEEE Software*, pages 63–72, January 1985.

[32] A. Borgida, S. Greenspan, and J. Mylopoulos. Knowledge representation as the basis for requirements specification. *IEEE Computer*, 18(4):82–91, April 1985.

[33] R. P. Boyle. Path analysis and ordinal data. *American Journal of Sociology*, 75:461–480, 1970.

[34] S. Bråten. *Dialogens vilkår i datasamfunnet (In Norwegian)*. Universitetsforlaget, 1983.

[35] F. P. Brooks Jr. No silver bullet. Essence and accidents of software engineering. In H. J. Kugler, editor, *Information Processing '86*, pages 1069–1076. North-Holland, 1986.

[36] T. Bryant and A. Evans. OO oversold: Those objects of obscure desire. *Information and Software Technology*, 36(1):35–42, January 1994.

[37] J. A. Bubenko jr. Information modelling in the context of systems development. In *IFIP Conference*, 1980. Invited paper.

[38] J. A. Bubenko jr. On concepts and strategies for requirements and information analysis. In *Information Modelling*, pages 125–169. Chartwell-Bratt Ltd., 1983.

[39] J. A. Bubenko jr. Problems and unclear issues with hierarchical business activity and data flow modelling. Technical Report 134, SYSLAB, Stockholm, June 1988.

[40] J. A. Bubenko jr, C. Rolland, P. Loucopoulos, and V. DeAntonellis. Facilitating fuzzy to formal requirements modelling. In *Proceedings of the First International Conference on Requirements Engineering (ICRE94)*, pages 154–157, Colorado Springs, USA, April 18-22 1994. IEEE Computer Society Press.

[41] C. V. Bullen and J. L. Bennett. Groupware in practice: An interpretation of work experiences. In C. Dunlop and R. Kling, editors, *Computerization and Controversy : Value Conflicts and Social Choices*, pages 257–287. Academic Press, 1991.

[42] G. Burrel and G. Morgan. *Sociological Paradigms and Organizational Analysis*. Heinemann, 1979.

[43] M. A. M. Capretz and M. Munro. Software configuration management issues in the maintenance of existing system. *Journal of Software Maintenance*, 6:1–14, 1994.

[44] J. M. Carey. Prototyping: Alternative systems development methodology. *Information and Software Technology*, 32(2):119–126, March 1990.

[45] E. Carmel, R. D. Whitaker, and J. F. George. PD and joint application design: A transatlantic comparison. *Communications of the ACM*, 36(4):40–48, June 1993.

[46] J. Carmo and A. Jones. Deontic database constraints and the characterisation of recovery. In Jones and Sergot [157], pages 56–85.

[47] Central Bureau of Statistics of Norway. *Statistical Yearbook of Norway*. Statistisk sentralbyrå, Oslo, Norway, 1992.

[48] P. Chaiyasut, G. Shanks, and P. M. C. Swatman. A computational architecture to support conceptual data model reuse by analogy. In Müller and Norman [229], pages 70–79.

[49] N. Chapin. Software maintenance: A different view. In *Proceedings of the National Computer Conference (AFIPS)*, pages 507–513, 1985.

[50] N. Chapin. Evidence on seperately organizing for software maintenance. In *Proceedings of the National Computer Conference (AFIPS)*, pages 517–522, 1987.

[51] P. B. Checkland. *Systems Thinking, Systems Practice*. John Wiley & Sons, 1981.

[52] P. B. Checkland and J. Scholes. *Soft Systems Methodology in Action*. Wiley, 1990.

[53] P. P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[54] L. Cholvy and F. Cuppens. Solving normative conflict by merging roles. In *Proceedings from the Fifth International Conference on Artificial Intelligence and law*, pages 201–209, University of Maryland, USA, May 21–24 1995. ACM Press.

[55] L. Chung. Dealing with security requirements during the development of information systems. In Rolland et al. [269], pages 234–251.

[56] J. Conklin and M. J. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, 1988.

[57] R. Conradi, G. M. Høydalsvik, and G. Sindre. A comparison of modelling frameworks for software processes and information systems. In *Proceedings of Third European Workshop on Software Process Technology (EWSPT'94)*, 1994.

[58] R. J. Costello and D-B. Liu. Metrics for requirements engineering. *Journal of Systems and Software*, 29(1):39–63, April 1995.

[59] B. Dahlbom. The idea that reality is socialy constructed. In Floyd et al. [91], pages 101–126.

[60] A. M. Davis. A comparison of techniques for the specification of external system behavior. *Communications of the ACM*, 31(9):1098–1115, September 1988.

[61] A. M. Davis. Object-oriented requirements to object-oriented design: An easy transition? *Journal of Systems and Software*, 30(1/2):151–159, July/August 1995.

[62] A. M. Davis, E. H. Bersoff, and E. R Comer. A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, 14(8):1453–1461, October 1988.

[63] A. M. Davis, S. Overmeyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos. Identifying and measuring quality in a software requirements specification. In *Proceedings of the First International Software Metrics Symposium*, pages 141–152, 1993.

[64] R. Davis and J. King. An overview of production systems. In E. W. Elcock and D. Michie, editors, *Machine Intelligence*, pages 300–332. 1977.

[65] G. de Michelis and M. A. Grasso. Situating conversations within the language/action perspective: The Milan conversation model. In *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work (CSCW'94)*, pages 89–100, Chapel Hill, North Carolina, USA, October 22-26 1994.

[66] J. K. Debenham and G. M. McGrath. The description in logic of large commercial data bases: A methodology put to the test. In *Proceedings of the Fifth Australian Computer Science Conference*, pages 12–21, 1982.

[67] S. M. Dekleva. Delphi study of software maintenance problems. In *Proceedings of the Conference on Sofware Maintenance (CSM'92)*, pages 10–17, 1992.

[68] S. M. Dekleva. The influence of the information systems development approach on maintenance. *MIS Quarterly*, pages 355–372, September 1992.

[69] S. M. Dekleva. Software maintenance : 1990 status. *Journal of Software Maintenance*, 4:233–247, 1992.

[70] DOD-STD-2167A. Departement of Defense, Washington D.C., 1987.

[71] J. Dietz. Integrating management of human and computer resources in task processing organizations: A conceptual view. In Nunamaker and Sprague [238], pages 723–733.

[72] J. L. G. Dietz and G. A. M. Widdershoven. A comparison of the lingusitic theories of Searle and Habermas as a basis for communication supporting systems. In R. P. van Riet and R. A. Meersman, editors, *Linguistic Instruments in Knowledge Engineering*, pages 121–130. Elsevier, 1992.

[73] F. Dignum, T. Kemme, W. Kreuzen, R. Weigand, and R. P. van de Riet. Constraint modelling using a conceptual prototyping language. *Data & Knowledge Engineering*, (2):213–254, 1987.

[74] F. Dignum and H. Weigand. Communication and deontic logic. In R. Wieringa and R. Feenstra, editors, *Working papers of the International Workshop on Information Systems - Correctness and Reuseability (IS-CORE '94)*, 1994.

[75] F. Dignum and H. Weigand. Modelling communication between cooperative systems. In Iivari et al. [143], pages 140–153.

[76] E. Downs, P. Clare, and I. Coe. *Structured Systems Analysis and Design Method: Application and Context*. Prentice Hall, 1988.

[77] K. G. Doyle, J. R. G. Wood, and A. T. Wood-Harper. Soft systems and systems engineering: On the use of conceptual models in information system development. *Journal of Information Systems*, 3(3):187–198, 1993.

[78] E. Dubois, P. Du Bois, and M. Petit. ALBERT: An agent-oriented language for building and elicitating requirements for real-time systems. In Nunamaker and Sprague [238], pages 713–722, Volume 4.

[79] S. Easterbrook. Domain modelling with hierarchies of alternative viewpoints. In *Proceedings of the IEEE International Symposium on Requirements Engineering (RE'93)*, pages 65–72, San Diego, USA, January 4-6 1993.

[80] P. Ehn. *Work-oriented Design of Computer Artifacts*. Arbetslivcentrum, 1989.

[81] A. Ekambaram, S. Gjertsen, J. P. Hjulstad, K. P. Humstad, Y. Johnsen, V. Larsen, and H. F. Stortiseth. C4 Cyber-conference coordination control. Technical report, IDT, NTH, Trondheim, Norway, 1994.

[82] M. Elden and M. Levin. Co-generative learning. Bringing participation into action research. In W. F. Whyte, editor, *Participative Action Research*. Sage, 1991.

[83] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., 1989.

[84] D. W. Embley, R. B. Jackson, and S. N. Woodfield. OO system analysis: Is it or isn't it? *IEEE Software*, 12(3):19–33, July 1995.

[85] M. Emery. The theory and practice of search conferences. In *Proceedings from the Einar Thorsrud Memorial Symposium and Workshop*, Oslo, Norway, June 15-19 1987.

[86] M. S. Feather. An implementation of bounded obligations. In *Proceedings of the Eight Knowledge-Based Software Engineering Conference (KBSE'93)*, pages 114–122, Chicago, USA, September 20-23 1993.

[87] M. S. Feather. Requirement reconnoitering at the juncture of domain and instance. In *Proceedings of the IEEE International Symposium on Requirements Engineering (RE'93)*, pages 73–76, San Diego, USA, January 4-6 1993.

[88] J. Floch. Supporting evolution and maintenance by using a flexible automatic code generator. In *17th International Conference on Software Engineering (ICSE'95)*, pages 211–219, Seattle, Washington, USA, April 23–30 1995.

[89] F. Flores, M. Graves, B. Hartfield, and T. Winograd. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems*, 6(2):153–172, April 1988.

[90] C. Floyd, F-M. Reisin, and G. Schmidt. STEPS to software development with users. In C. Ghezzi and J. A. McDermid, editors, *2nd European Software Engineering Conference (ESEC'89)*, pages 48–63, University of Warwick, Coventry, England, September 1989.

[91] C. Floyd, H. Züllighoven, R. Budde, and R. Keil-Slawik, editors. *Software Development and Reality Construction*. Springer Verlag, 1991.

[92] F. Foffani. Survey on quality in software maintenace in Italy. Newsletter, Center for Software Maintenance, Durham, England, May 1992.

[93] J. R. Foster. Program lifetime: A vital statistics for maintenance. In *Proceeding of the Conference on Software Maintenance (CSM'91)*, pages 98–103, 1991.

[94] C. Francalanci and B. Pernici. View integration: A survey of current developments. Technical Report 93-053, Politecnico de Milano, Milan, Italy, 1993.

[95] Personal communication with the FRISCO task group, March 1995.

[96] J. Galtung. *Theory and Method of Social Research*. Universitetsforlaget, Oslo, Norway, 1967.

[97] C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice-Hall, 1979.

[98] D. G. Gause and G. M. Weinberg. *Exploring Requirements: Quality before Design*. 1989.

[99] M. R. Genesereth and S. T. Ketchpel. Software agents. *Communication of the ACM*, 37(7):48–53, July 1994.

[100] M. Genuchthen. Why is software late? An experimental study of reasons for delay in software development. *IEEE Transactions on Software Engineering*, 17(6):582–590, June 1991.

[101] W. W. Gibbs. Software's chronic crisis. *Scientific American*, 271(3):86–95, September 1994.

[102] R. Gjersvik. *The Construction of Information Systems in Organization: An Action Research Project on Technology, Organizational Closure, Reflection, and Change*. PhD thesis, ORAL, NTH, Trondheim, Norway, 1993.

[103] R. L. Glass. We have lost our way. *Journal of Systems and Software*, 18(2):111–112, May 1992.

[104] R. L. Glass and I. Vessey. Contemporary application-domain taxonomies. *IEEE Software*, 12(3):63–76, July 1995.

[105] B. C. Glasson. Model of system evolution. *Information and Software Technology*, 31(7):351–356, September 1989.

[106] J. A. Goguen and C. Linde. Techniques for requirements elicitation. In *Proceedings of the IEEE International Symposium on Requirements Engineering (RE'93)*, pages 152–164, 1993.

[107] G. Goldkuhl and K. Lyytinen. A language action view of information systems. In *Proceedings of the International Conference on Information Systems (ICIS'82)*, pages 13–29. 1982.

[108] G. Goldkuhl and K. Lyytinen. Information systems specification as rule reconstruction. In Th. M. A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 79–94. North-Holland, 22-24 August 1983.

[109] O. C. Z. Gotel and A. Finkelstein. An analysis of the requirements tracability problem. In *Proceedings of the First International Conference on Requirements Engineering (ICRE'94)*, pages 94–101, Colorado Springs, USA, April 18-22 1994. IEEE.

[110] E. S. Greenberg. The consequences of worker participation: A clarification of the theorethical litterature. *Social Science Quarterly*, 56(2), 1975.

[111] E. G. Guba and Y. S. Lincoln. *Fourth Generation Evaluation*. Sage, 1989.

[112] J. P. Guildford. *Fundamental Statistics in Psychology and Education*. McGraw-Hill, 1978.

[113] T. Guimaraes. Managing application program maintenance expenditure. *Communications of the ACM*, 26(10):739–746, 1983.

[114] J. A. Gulla. *Explanation Generation in Information Systems Engineering*. PhD thesis, IDT, NTH, Trondheim, Norway, September 1993.

[115] J. A. Gulla and O. I. Lindland. Modelling cooperative work for workflow management. In Wijers et al. [330], pages 53–65.

[116] J. A. Gulla, O. I. Lindland, and G. Willumsen. PPP - An integrated CASE environment. In Andersen et al. [5], pages 194–221.

[117] M. R. Gustafsson, T. Karlsson, and J. A. Bubenko jr. A declarative approach to conceptual information modelling. In Olle et al. [243], pages 93–142.

[118] J. Habermas. *The Theory of Communicative Action*. Beacon Press, 1984.

[119] J. Hagelstein. A declarative approach to information systems requirements. *Knowledge Based Systems*, 1(4):211–220, 1988.

[120] G. F. Håland. SIMSPEC - Simulating a PPP specification. Master's thesis, IDT, NTH, Trondheim, Norway, 1991.

[121] D. P. Hale, D. A. Haworth, and S. Sharpe. Empirical software maintenance studies during the 1980s. In *Proceedings of the Conference on Software Maintenance (CSM'90)*, pages 118–123. IEEE Computer Society Press, 1990.

[122] C. Hardy, S. Stobart, B. Thompson, and H. Edwards. A comparison of the results of two surveys on software development and the role of CASE in the UK. In Müller and Norman [229], pages 234–238.

[123] C. Hardy, B. Thompson, and H. Edwards. A preliminary survey of method use in the UK. Technical Report 94-12, School of Computing and Information, University of Sunderland, UK, May 1994.

[124] D. Harel. Statecharts : A visual formalism for complex systems. *Science of Computer Programming*, (8):231–274, 1987.

[125] D. Harel. Biting the silver bullet: Toward a brighter future for system development. *IEEE Computer*, January 1992.

[126] D. Harel, H. Lachover, A. Naamed, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: a working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[127] W. Harrison and H. Ossher. Subject-oriented programming (a critique of pure objects). In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93)*, pages 411–428, 1993.

[128] F. Heller. Participation and competence: A necessary relationship. In R. Russel and V. Rus, editors, *International Handbook of Participation in Organizations*, pages 265–281. 1991.

[129] B. Henderson-Sellers and J. M. Edwards. The object-oriented systems life cycle. *Communications of the ACM*, 33(9):142–159, September 1990.

[130] A. Henne. Information systems maintenance - Problems or opportunities? In *Proceedings of Norsk Informatikk Konferanse 1992 (NIK'92)*, pages 91–104, Tromsø, Norway, 1992.

[131] H. Herrestad and C. Krogh. Obligations directed from bearers to counterparties. In *Proceedings from the Fifth International Conference on Artificial Intelligence and Law*, pages 210–218, University of Maryland, USA, May 21–24 1995. ACM Press.

[132] J. Hewett and T. Durham. CASE: The Next Steps. Technical report, OVUM, 1989.

[133] R. A. Hirschheim. A participative approach to implementing office automation. In *Proceedings from the Joint International Symposium on Information Systems*, pages 306–329, Sydney, Australia, April 1984.

[134] R. A. Hirschheim and H. K. Klein. Four paradigms of information systems development. *Communications of the ACM*, 32(10):pages 1199–1216, October 1989.

[135] M. A. Hitt, R. D. Middlemist, and R. L. Mathis. *Management*. West, 1986.

[136] P. Holm. The COMMODIOUS method: Communication modelling as an aid to illustrate the organizational use of software. In *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering (SEKE'94)*, pages 10–19, Jurmala, Latvia, June 21-23 1994. Knowledge Systems Institute.

[137] G. M. Høydalsvik and G. Sindre. On the purpose of object-oriented analysis. In A. Paepcke, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93)*, pages 240–255. ACM Press, September 1993.

[138] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[139] J. Iivari. Hierarchical spiral model for information system and software development. Part 2: Design process. *Information and Software Technology*, 32(7):450–458, September 1990.

[140] J. Iivari. Hierarchical spiral model for information system and software development. Part 1: Theoretical background. *Information and Software Technology*, 32(6):386–399, July/August 1990.

[141] J. Iivari. Object-orientation as structural, functional and behavioural modelling: A comparison of six methods for object-oriented analysis. *Information and Software Technology*, 37(3):155–163, 1995.

[142] J. Iivari and E. Koskela. The PIOCO model for information systems design. *MIS Quarterly*, pages 401–419, September 1987.

[143] J. Iivari, K. Lyytinen, and M. Rossi, editors. *Proceedings of the 7th International Conference on Advanced Information Systems Engineering (CAiSE'95)*, Jyväskylä, Finland, June 12-16 1995. Springer Verlag.

[144] S. Ilstad. *Survey-metoden: En veiledning i utvalgsundersøkelser (In Norwegian)*. Tapir, 1989.

[145] ISO Standard 9000-3 Quality Management and Quality Assurance Standards. Part 3: Guidelines for the application of ISO9001 to the development, supply and maintenance of software. ISO, 1991.

[146] S. Jacobs, M. Jarke, and K. Pohl. Report on the first international IEEE symposium on requirements engineering (RE'93) San Diego, January 4–6 1993. *Automated Software Engineering*, 1(1):129–132, 1994.

[147] M. Jarke, J. A. Bubenko jr, C. Rolland, A. Sutcliffe, and Y. Vassiliou. Theories underlying requirements engineering: An overview of NATURE at genesis. In *Proceedings of the IEEE International Symposium on Requirements Engineering (RE'93)*, pages 19–31, 1993.

[148] M. Jarke, J. Mylopoulos, J. W. Schmidt, and Y. Vassiliou. DAIDA: An environment for evolving information systems. *ACM Transactions on Information Systems*, 10(1):1–50, 1992.

[149] A. M. Jenkins, J. D. Naumann, and J. C. Wetherbe. Empirical investigations of systems development practices and results. *Information Manager*, 7:73–82, 1984.

[150] Y. Johnsen and T. Karlsen. P3app. Technical report, IDT, NTH, Trondheim, Norway, 1995.

[151] W. L. Johnson and M. S. Feather. Using evolution transformations to construct specifications. In Lowry and McCartney [204], pages 65–91.

[152] W. L. Johnson, M. S. Feather, and D. R. Harris. Representation and presentation of requirements knowledge. *IEEE Transactions on Software Engineering*, 18(10):853–869, October 1992.

[153] A. J. I. Jones. Deontic logic and legal knowledge representation. *Ratio Juris*, 1990.

[154] A. J. I. Jones. Towards a formal theory of defeasible deontic conditionals. *Annals of Mathematics and Artificial Intelligence*, 1993.

[155] A. J. I. Jones and I. Pörn. Ideality, sub-ideality and deontic logic. *Synthese*, 65:275–290, 1985.

[156] A. J. I. Jones and I. Pörn. "ought" and "must". *Synthese*, 66:89–93, 1986.

[157] A. J. I. Jones and M. Sergot, editors. *DEON'94: Second International Workshop on Deontic Logic in Computer Science*, Oslo, Norway, January 6-8 1994. TANO.

[158] M. Jørgensen. *Empirical studies of Software Maintenance*. PhD thesis, Department of Informatics, University of Oslo, Oslo, Norway, 1994.

[159] M. Jørgensen. An empirical study of software maintenance tasks. *Journal of Software Maintenance*, 7(1):27–48, 1995.

[160] M. Jørgensen. The quality of questionnaire based software maintenace studies. *ACM SIGSOFT Software Engineering Notes*, 20(1):71–73, January 1995.

[161] E.-A. Karlsson (ed.). *Software Reuse: A Holistic Approach*. John Wiley & Sons, 1995.

[162] M. Keil and E. Carmel. Customer-developer links in software development. *Communications of the ACM*, 38(5), May 1995.

[163] S. J. H. Kent, T. S. E. Maibaum, and W. J. Quirk. Formally specifying temporal constraints and error recovery. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 208–215, San Diego, USA, January 4-6 1993.

[164] H. Klein and K. Lyytinen. Towards a new understanding of data modelling. In Floyd et al. [91], pages 203–217.

[165] R. Kling. Social analysis of computing: Theoretical perspectives in recent empirical research. *ACM Computing Survey*, 12(1):61–110, March 1980.

[166] W. Kozaczynski and A. Kuntzmann-Combelles. What it takes to make OO work. *IEEE Software*, 10(1):20–23, January 1993.

[167] J. Krogstie. A quasi-natural language for formal representation of rules in TEMPORA. Project report, IDT, NTH, Trondheim, Norway, May 8 1990.

[168] J. Krogstie. Conceptual modeling in Tempora. Master's thesis, IDT, NTH, Trondheim, Norway, 1991.

[169] J. Krogstie. CASE - Computer aided system evolution? Workshop Reports. Sixth International Workshop on Computer-Aided Software Engineering (CASE'93), July 19–23 1993.

[170] J. Krogstie. Information systems development and maintenance in Norway: A survey investigation. In *Norsk konferanse for organisasjoners bruk av informasjonsteknologi (NOKOBIT'94)*, pages 1–22, Bergen, Norway, June 16–17 1994.

[171] J. Krogstie. Survey investigation: Development and maintenance of information systems in Norway. Technical Report 0802-6394 6/94, IDT, NTH, Trondheim, Norway, July 30 1994.

[172] J. Krogstie. Goal-oriented modeling of information systems. In *Proceedings of the Seventh International Conference on Computing and Information (ICCI'95)*, pages 983–1007, Peterborough, Canada, July 5–8 1995.

[173] J. Krogstie. On the distinction between functional development and functional maintenance. To be published in Journal of Software Maintenance, 1995.

[174] J. Krogstie. Towards a terminology for information systems? In *Preprint of IFIP8.1 ISCO3 conference*, Marburg, Germany, March 28-30 1995.

[175] J. Krogstie. Use of development methodology and CASE-tools in Norway: Results from a survey. In H. A. Müller and R. J. Norman, editors, *Proceedings of the Seventh International Workshop on Computer-Aided Software Engineering (CASE'95)*, pages 239–248, Toronto, Canada, July 10-14 1995.

[176] J. Krogstie. Use of methods and CASE-tools in Norway: Results from a survey. Invited to special issue on CASE in Journal of Automated Software Engineering, 1996.

[177] J. Krogstie, O. I. Lindland, and G. Sindre. Defining quality aspects for conceptual models. In *Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO3): Towards a Consolidation of Views*, Marburg, Germany, March 28-30 1995.

[178] J. Krogstie, O. I. Lindland, and G. Sindre. Towards a deeper understanding of quality in requirements engineering. In Iivari et al. [143], pages 82–95.

[179] J. Krogstie, P. McBrien, R. Owens, and A. H. Seltveit. Information systems development using a combination of process and rule based approaches. In Andersen et al. [5], pages 319–335.

[180] J. Krogstie and G. Sindre. Extending a temporal rule language with deontic operators. In *Proceedings from the 6th International Conference on Software Engineering and Knowledge Engineering (SEKE'94)*, pages 314–321. IEEE, June 21-23 1994.

[181] J. Krogstie and A. Sølvberg. Software maintenance in Norway: A survey investigation. In Müller and Georges [228], pages 304–313. Received "Best Paper Award".

[182] C. H. Kung. A temporal framework for database specification and verification. In *Proceedings of the tenth International Conference on Very Large Data Bases (VLDB'84)*, pages 91–99, Singapore, August 27-31 1984.

[183] C. H. Kung. Conceptual modeling in the context of software development. *IEEE Transactions on Software Engineering*, 15(10):1176–1187, October 1990.

[184] R. J. Kusters and G. M. Wijers. On the practical use of CASE-tools. Results of a survey. In Lee et al. [187], pages 2–10.

[185] M. Kyng. Designing for cooperation: Cooperating for design. *Communications of the ACM*, 34(12):65–73, December 1991.

[186] P. J. Layzell and L. Macauley. An investigations into software maintenance - perception and practices. *Software Maintenace: Research and Practice*, 6:105–119, 1994.

[187] H. Y. Lee, T. Reid, and S. Jarzabek, editors. *Proceedings of the 6th International Workshop on Computer-Aided Software Engineering (CASE'93)*, Singapore, July 1993. IEEE Computer Society Press.

[188] R. M. Lee. Bureaucraties as deontic systems. *ACM Transactions on Office Information Systems*, 6(2):87–108, 1988.

[189] M. M. Lehman. On understanding laws, evolution, and conservation in the large-program life-cycle. *Journal for Systems and Software*, 1:213–221, 1980.

[190] J. C. S. P. Leite and P. A. Freeman. Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering*, 17(12):1253–1269, December 1991.

[191] P. J. Lewis. Linking soft systems methodology with data-focused information systems development. *Journal of Information Systems*, 3:169–186, 1993.

[192] X. Li. What's so bad about rule-based programming? *IEEE Software*, 8(5):103,105, September 1991.

[193] B. P. Lientz and E. B. Swanson. *Software Maintenance Management*. Addison Wesley, 1980.

[194] Y. S. Lincoln and E. G. Guba. *Naturalistic Inquiry*. Sage, 1985.

[195] P. Lindgren ed. A framework of information systems concepts. Technical Report Interrim report, FRISCO, May 1990.

[196] O. I. Lindland. *A Prototyping Approach to Validation of Conceptual Models in Information Systems Engineering*. PhD thesis, IDT, NTH, Trondheim, Norway, May 1993.

[197] O. I. Lindland and J. Krogstie. Transformations in CASE tools - A compiler view. In Lee et al. [187], pages 287–296.

[198] O. I. Lindland and J. Krogstie. Validating conceptual models by transformational prototyping. In Rolland et al. [269], pages 165–183.

[199] O. I. Lindland, G. Sindre, and A. Sølvberg. Understanding quality in conceptual modelling. *IEEE Software*, pages 42–49, April 1994.

[200] T. L. Lo. The house of quality and service management. *CMG Transactions*, pages 39–46, Fall 1994.

[201] G. Longworth. Specification and development of software systems. Technical report, GMD, 1985.

[202] P. Loucopoulos, P. McBrien, F. Schumacker, B. Theodoulidis, V. Kopanas, and B. Wangler. Integrating database technology, rule-based systems and temporal reasoning for effective information systems: The TEMPORA paradigm. *Journal of Information Systems*, 1:129–152, 1991.

[203] K. W. Løvseth. TEMOCCA - TEMPORA modelling concepts - A case study. Master's thesis, IDT, NTH, Trondheim, Norway, May 31 1990.

[204] M. R. Lowry and R. D. McCartney, editors. *Automating Software Design*, California, USA, 1991. The MIT Press.

[205] P. H. Loy. A comparison of object-oriented and stuctured development methods. *ACM SIGSOFT Software Engineering Notes*, 15(1):44–48, January 1990.

[206] H. C. Lucas Jr. *Why Information Systems Fail*. Columbia University Press, 1975.

[207] M. Lundeberg. The ISAC approach to specification of information systems and its application to the organization of an IFIP working conference. In *Information Systems Design Methodologies: A Comparative Review*, pages 173–234. North-Holland, 1982.

[208] K. Lyytinen. A taxonomic perspective of information systems development: Theoretical constructs and recommendations. In R. J. Boland Jr and R. A. Hirschheim, editors, *Critical Issues in Information Systems Research*, chapter 1, pages 3–41. John Wiley & Sons, 1987.

[209] K. Lyytinen, P. Kerola, J. Kaipala, S. Kelly, J. Lehto, H. Liu, P. Marttiin, H. Oinas-Kukkonen, J. Prihonen, M. Rossi, K. Smolander, V-P. Tahvanainen, and J-P. Tolvanen. MetaPHOR: Metamodelling, principles, hypertext, objects, and repositories. Technical Report TR-7, University of Jyväskylä, Department of Computer Science and Information Systems, Jyväskylä, Finland, November 1994.

[210] K. Lyytinen and E. Lehtinen. Discourse analysis as an information systems specification method. In *Report of the 7th. Scandinavian Research Seminar on Systemeering*, pages 146–200, Helsinki Business School, Finland, 1984.

[211] L. Macauley. Requirements capture as a cooperative activity. In *Proceedings of the First Symposium on Requirements Engineering (RE'93)*, pages 174–181, 1993.

[212] T. S. E. Maibaum. Temporal reasoning over deontic specifications. In J. J. C. Meyer and R. J. Wieringa, editors, *Deontic Logic in Computer Science: Normative Systems Specifications*, pages 141–202. John Wiley & Sons, 1993.

[213] M. A. Marsan et al, editor. *Proceeding of the International workshop on Timed Petri Nets*, Torino, Italy, 1985. IEEE Computer Society Press.

[214] P. McBrien, M. Niezette, D. Pantazis, A. H. Seltveit, U. Sundin, B. Theodoulidis, G. Tziallas, and R. Wohed. A rule language to capture and model business policy specifications. In Andersen et al. [5], pages 307–318.

[215] P. Mcbrien and A. H. Seltveit. Coupling process models and business rules. In Sølvberg et al. [291], pages 201–217.

[216] P. McBrien, A. H. Seltveit, and B. Wangler. An entity-relationship model extended to describe historical information. In *Proceedings of CISMOD'92*, Bangalore, India, July 1992.

[217] L. T. McCarty. A language for legal discourse: Basic features. In *The Second International Conference on Artificial Intelligence and Law*, pages 180–189, June 25–28 1989.

[218] D. McCracken and M. Jackson. Life cycle concept considered harmful. *ACM SIGSOFT Software Engineering Notes*, 7(2):29–32, April 1982.

[219] J. D. McGregor and T. D. Korson. Integrated object-oriented testing and development processes. *Communications of the ACM*, 37(9), 1994.

[220] P. McNamara. Doing well enough: Toward a logic of common sense morality. In Jones and Sergot [157], pages 165–197.

[221] R. Medina-Mora, T. Winograd, R. Flores, and F. Flores. The action workflow approach to workflow management technology. In *Proceedings of CSCW'92*, 1992.

[222] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Hemel Hempstead, 1988.

[223] J-J. C. Meyer and R. J. Wieringa, editors. *DEON'91: First International Workshop on Deontic Logic in Computer Science*, Amsterdam, Holland, December 11-13 1991. Vrije Universiteit Amsterdam.

[224] N. H. Minsky and A. Lockman. Extending authorization by adding obligations to permissions. Technical report, Computer Science Laboratory, Rutgers University, New Brunswick, Canada, January 1985.

[225] D. L. Moody and G. G. Shanks. What makes a good data model? Evaluating the quality of entity relationship models. In *Proceedings of the 13th International Conference on the Entity-Relationship Approach (ER'94)*, pages 94–111, Manchester, England, 1994.

[226] A. I. Mørch. Designing for radical tailorability: Coupling artifact and rationale. *Knowledge-Based Systems*, 7(4):253–264, December 1994.

[227] G. Morgan. *Images of Organization.* Sage Publications, Inc., 1986.

[228] H. A. Müller and M. Georges, editors. *Proceedings of the International Conference on Software Maintenance (ICSM'94)*. IEEE COmputer Society Press, September 19-23 1994.

[229] H. A. Müller and R. J. Norman, editors. *Proceedings of the Seventh International Workshop on Computer-Aided Software Engineering (CASE'95)*, Toronto, Canada, July 10-14 1995. IEEE Computer Society Press.

[230] E. Mumford. Designing human systems - the ETHICS method. Technical report, Manchester Business School, Cheshire, England, 1983.

[231] E. Mumford. Participation - from Aristotle to today. In Th. M. A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 95–104. North-Holland, 22-24 August 1983.

[232] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using non-functional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, June 1992.

[233] V. K. Narayanan and R. Nath. *Organization Theory: A Strategic Approach*. Irwin, 1993.

[234] N. H. Nie, C. H. Hull, J. G. Jenkins, K. Steinbrenner, and D. H. Bent. *Statistical Package for Social Science*. McGraw-Hill, 1975.

[235] O. Noreide. Actor modelling in PPP. Technical report, IDT, NTH, Trondheim, Norway, 1995.

[236] M. J. Norusis. *SPSS for Windows: Base System User's Guide*. SPSS Inc., Chicago, Illinois, USA, 1992.

[237] J. T. Nosek and P. Palvia. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance*, 2:157–174, 1990.

[238] J. F. Nunamaker and R. H. Sprague, editors. *Proceedings of the Twenty-seventh Annual Hawaii International Conference on Systems Sciences (HICCS'27)*, Maui, Hawaii, US, January 4–7 1994. IEEE Computer Society Press.

[239] J. F. Nunamaker and R. H. Sprague, editors. *Proceedings of the Twenty-eight Annual Hawaii International Conference on Systems Sciences (HICCS'28)*, Maui, Hawaii, US, January 1995. IEEE Computer Society Press.

[240] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, October 1994.

[241] A. Olivé. A comparison of the operational and deductive approaches to conceptual information systems modelling. In H. J. Kugler, editor, *Information Processing '86*. North-Holland, 1986.

[242] T. W. Olle, J. Hagelstein, I. G. MacDonald, C. Rolland, H. G. Sol, F. J. M. van Assche, and A. A. Verrijn-Stuart. *Information Systems Methodologies*. Addison-Wesley, 1988.

[243] T. W. Olle, H. G. Sol, and A. A. Verrijn-Stuart, editors. *Information Systems Design Methodologies: A Comparative Review*. North-Holland, 1982.

[244] A. L. Opdahl. *Performance Engineering during Information Systems Development*. PhD thesis, IDT, NTH, Trondheim, Norway, 1992.

[245] A. L. Opdahl and G. Sindre. A taxonomy for real-world modelling concepts. *Information Systems*, 19(3):229–241, April 1994.

[246] A. L. Opdahl and G. Sindre. Facet models for problem analysis. In Iivari et al. [143], pages 54–67.

[247] A. L. Opdahl and G. Sindre. Representing real-world processes. In Nunamaker and Sprague [239].

[248] J. W. Orlikowski and D. C. Gash. Technological frames: Making sense of information technology in organizations. *ACM Transactions on Information Systems*, 12(2):174–207, 1994.

[249] OVUM evaluates CASE products. Technical report, OVUM, September 1993. Including extensions as of February 1994.

[250] P. Pagin. *Ideas for a Theory of Rules*. PhD thesis, Stockholm University, Stockholm, Sweden, 1987.

[251] Y. Park and D. Ramjisingh. Software component base for reuse in functional program development. In *Proceedings of the Seventh International Conference on Computing and Information (ICCI'95)*, pages 1022–1039, July 5–8 1995.

[252] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. Capability maturity model, version 1.1. *IEEE Software*, 10(4):pages 18–27, July 1993.

[253] J. Peckham and F. Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):pages 153–190, September 1988.

[254] B. Pernici and C. Rolland. Automatic tools for designing office information systems (TODOS). Research report 813, ESPRIT, 1990.

[255] D. E. Perry. Dimensions of software evolution. In Müller and Georges [228], pages 296–303.

[256] T. Peters. *Thrieving in Chaos*. Billings & Sons, 1987.

[257] M. Petre. Why looking isn't always seeing. Readership skills and graphical programming. *Communications of the ACM*, 38(6):pages 33–44, June 1995.

[258] C. A. Petri. Kommunikation mit automaten (In German). *Schriften des Rheinisch-Westfalischen Institut für Instrumentelle Mathematik an der Universität Bonn*, (2), 1962.

[259] T. Pettersen. Rule capturing and structuring in TEMPORA. Master's thesis, IDT, NTH, Trondheim, Norway, January 27 1992.

[260] D. Phan, D. Vogel, and J. F. Nunamaker. The search for the perfect project management. *Computerworld*, pages 95–100, 1988.

[261] K. Pohl. The three dimensions of requirements engineering: A framework and its applications. *Information Systems*, 19(3):243–258, April 1994.

[262] K. Pohl and P Haumer. Hydra: A hypertext model for structuring informal requirements representations. In P. Peters and K. Pohl, editors, *REFSQ'95*, 1995.

[263] R. Prieto-Diaz. Status report: Software reuseability. *IEEE Software*, pages 61–66, May 1993.

[264] C. Quer and A. Olivé. Object interaction in object-oriented deductive conceptual models. In Rolland et al. [269], pages 374–396.

[265] M. R. Raabel. User interface specification language for PPP. Master's thesis, IDT, NTH, Trondheim, Norway, 1993.

[266] B. Ramesh and M. Edwards. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Transactions on Software Engineering*, 18(6):498–510, June 1992.

[267] T. Reenskaug, P. Wold, and O. A. Lehne. *Working with Objects*. Manning/Prentice Hall, 1995.

[268] H. Rittel. On the planning crisis: Systems analysis of the first and second generations. *Bedriftsøkonomen*, 34(8), 1972.

[269] C. Rolland, F. Bodart, and C. Cauvet, editors. *Proceedings of the 5th International Conference on Advanced Information Systems Engineering (CAiSE'93)*, Paris, France, June 8-11 1993. Springer Verlag.

[270] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings WESCON*, August 1970.

[271] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[272] M. Ryan. Towards specifying norms. In Meyer and Wieringa [223], pages 335–346.

[273] Y. U. Ryu. Conditional deontic logic augmented with defeasible reasoning. *Data & Knowledge Engineering*, 16(1):pages 73–91, July 1995.

[274] *Rett sats: Kartlegningskonferanser som utgangspunkt for bedriftsutvikling (In Norwegian)*. Senter for bedre arbeidsliv, 1991.

[275] A-W. Scheer and A. Hars. Extending data modelling to cover the whole enterprise. *Communications of the ACM*, 35(9):166–171, September 1992.

[276] D. Schuler and A. Namioka. *Participatory design: Principles and Practices*. Lawrence Erlbaum, 1993.

[277] F. Schumacker. Towards an execution mechanism for TEMPORA. Technical Report E2469/ULG/T3.2/1/2, ULG, Liége, Belgium, March 15 1990.

[278] J. R. Searle. *Speech Acts*. Cambridge University Press, 1969.

[279] J. R. Searle. *Expression and Meaning*. Cambridge University Press, 1979.

[280] J. R. Searle and D. Vanderveken. *Foundations of Illocutionary Logic*. Cambridge University Press, 1985.

[281] A. H. Seltveit. An abstraction-based rule approach to large-scale information systems development. In Rolland et al. [269], pages 328–351.

[282] A. H. Seltveit. *Complexity Reduction in Information Systems Modelling*. PhD thesis, IDT, NTH, Trondheim, Norway, 1994.

[283] J. A. Senn. *Analysis & Design of Information Systems*. McGraw-Hill, 1989.

[284] Y. Shoham. Agent oriented programming: An overview of the framework and summary of recent research. In M. Masuch and L. Pólos, editors, *Knowledge Representation and Reasoning under Uncertainty: Logic at Work*, pages 123–129. Springer Verlag, 1994.

[285] J. Siddiqi. Challenging universal truths of requirements engineering. *IEEE Software*, pages 18–19, March 1994.

[286] G. Sindre. *HICONS: A General Diagrammatic Framework for Hierarchical Modelling*. PhD thesis, IDT, NTH, Trondheim, Norway, 1990. NTH report 1990:44, IDT report 1990:31.

[287] G. Sindre and J. Krogstie. Process heuristics to achieve requirements specification of feasible quality. In K. Pohl and P. Peters, editors, *Second International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'95)*, pages 92–103, Jyväskylä, Finland, June 12-13 1995.

[288] B. Singh and G. L. Rein. Role interaction nets (RINs):. Technical Report CT-083-92, MCC, Austin, Texas, July 22 1992.

[289] J. Slonim. OO in the real world - success or latest fashion? In Müller and Georges [228], pages 440–441.

[290] A. Sølvberg. A contribution to the definition of concepts for expressing users' information systems requirements. In P. P. Chen, editor, *Entity-Relationship Approach to Systems Analysis and Design*. North-Holland, 1980.

[291] A. Sølvberg, J. Krogstie, and A. H. Seltveit, editors. *Proceedings of the IFIP8.1 WC on Information Systems Development for Decentralized Organizations (ISDO'95)*, Trondheim, Norway, 21-23 August 1995. Chapman & Hall.

[292] A. Sølvberg and C. H. Kung. On structural and behaviour modelling of reality. In T. B. Steele and R. Meersman, editors, *Database Semantics*. North-Holland, 1986.

[293] A. Sølvberg and C. H. Kung. *Information Systems Engineering*. Springer-Verlag, 1993.

[294] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison Wesley, 1983.

[295] R. Stamper. Semantics. In R. J. Boland Jr and R. A. Hirschheim, editors, *Critical Issues in Information Systems Research*, pages 43–78. John Wiley & Sons, 1987.

[296] S. C. Stobart, J. B. Thompson, and P. Smith. Use, problems, benefits, and future direction of computer-aided software engineering in the United Kingdom. *Information and Software Technology*, 33(9):629–636, November 1991.

[297] A. G. Sutcliffe and N. A. M. Maiden. Bridging the requirements gap: Policies, goals and domains. In *Proceedings of the Seventh International Workshop on Software Specification and Design (IWSSD-7)*, pages 52–55, Redondo Beach, USA, December 6-7 1993.

[298] E. B. Swanson. The dimensions of maintenance. In *Proceedings of the Second International Conference on Software Engineering (ICSE'76)*, pages 492–497, San Francisco, USA, August 13-15 1976.

[299] E. B. Swanson and C. M. Beath. *Maintaining Information Systems in Organizations.* Wiley Series in Information Systems. John Wiley & Sons, 1989.

[300] W. R. Swartout and R. Balzer. On the inevitable intertwining of specification and implementation. *Communications of the ACM*, 25(7):438–440, July 1982.

[301] T. Tamai and Y. Torimitsu. Software lifetime and its evolution process over generations. In *Proceedings of the 1992 Conference on Software Maintenance (CSM 1992)*, pages 63–69. IEEE, 1992.

[302] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):61–79, January 1988.

[303] T. Taylor and T. A. Standish. Initial thoughts on rapid prototyping techniques. *ACM SIGSOFT Software Engineering Notes*, 7(5):160–166, December 1982.

[304] Tempora: Integrating database technology, rule-based systems and temporal reasoning for effective software. Technical Report ESPRIT Project 2469, Technical Annex, Tempora Consortium, October 17 1988.

[305] TEMPORA Consortium. *Tempora Project Manual*, August 1992.

[306] The Sweden Post case study. Technical report, TEMPORA Consortium, 1991.

[307] Tempora final review. Technical report, Tempora Consortium, 1994.

[308] I. Thomas and B. A. Nejmeh. Definitions of tool integration for environments. *IEEE Software*, 9(2):29–35, March 1992.

[309] K. Thoresen, V. Keul, J. Bing, A. Pape, and T. C. Paper. *Omstilling med IT (In Norwegian)*. NKS-forlaget, 1992.

[310] C. Tomlinson and M. Scheevel. Concurrent programming. In W. Kim and F. H. Lochovsky, editors, *Object-oriented Concepts, Databases and Applications*. Addison-Wesley, 1989.

[311] J. A. Turner. Observations on the use of behavioral models in information systems research and practice. *Information Manager*, 5(3):207–213, 1982.

[312] W. Twining and D. Miers. *How to do Things with Rules*. Weidenfeld and Nicholson, 1982.

[313] F. van Assche, P. Layzell, P. Loucopoulos, and G. Speltincx. Information systems development: A rule-based approach. *Knowledge Based Systems*, 1(4):227–234, September 1988.

[314] V. E. van Reijswoud and N. B. J. van der Rijst. Modelling business communication as a foundation for business process redesign: A case of production logistics. In Nunamaker and Sprague [239], pages 841–851.

[315] V. van Swede and H. van Vliet. Consistent development: Results of a first empirical study of the relation between project scenario and success. In Wijers et al. [330], pages 80–93.

[316] T. F. Verhoef and A. H. M. Hofstede. Feasibility of flexible information modelling support. In Iivari et al. [143], pages 168–185.

[317] R. Veryard and J. Dobson. Third order requirements engineering: Specification, change, and identity. In K. Pohl and P. Peters, editors, *REFSQ'95*, 1995.

[318] I. Vessey and S. A. Conger. Requirements specification: Learning object, process, and data methodologies. *Communications of the ACM*, 37(5):102–113, May 1994.

[319] I. Vessey and R. Weber. Some factors affecting programming repair maintenance. *Communications of the ACM*, 26(2):128–134, February 1983.

[320] M. Vestli, I. Nordbø, and A. Sølvberg. Modeling control in rule-based systems. *IEEE Software*, pages 77–81, March 1994.

[321] G. H. Von Wright. *An Essay in Deontic Logic and the General Theory of Action*. North-Holland, 1972.

[322] R. Vonk. *Prototyping — The effective use of CASE technology*. Prentice Hall, 1990.

[323] Y. Wand and R. Weber. On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems*, 3(4):217–237, 1993.

[324] B. Wangler, R. Wohed, and S-E. Öhlund. Business modelling and rule capture in a CASE environment. In *Proceedings of the Fourth Workshop on The Next Generation of CASE Tools*, Twente, The Netherlands, 1993.

[325] P. T. Ward. The transformation schema: An extension of the data flow diagram to represent control and timing. *IEEE Transactions on Software Engineering*, 12(2):198–210, February 1986.

[326] P. Wegner. Dimension of object-based language design. In *Proceedings of the Conference on Object Oriented Programming, Systems, and Languages (OOPSLA'87)*, pages 168–182. ACM Press, 1987.

[327] R. Wieringa. Three roles of conceptual models in information system design and use. In E. Falkenberg and P. Lindgren, editors, *Information System Concepts: An In-Depth Analysis*, pages 31–51. North-Holland, 1989.

[328] R. J. Wieringa and J-J. C. Meyer. Applications of deontic logic in computer science: A concise overview. In Meyer and Wieringa [223], pages 15–41.

[329] R. J. Wieringa, J-J. C. Meyer, and H. Weigand. Specifying dynamic and deontic integrity constraints. *Data and Knowledge Engineering*, 4:157–189, 1989.

[330] G. Wijers, S. Brinkkemper, and T. Wasserman, editors. *Proceedings of the 6th International Conference on Advanced Information Systems Engineering (CAiSE'94)*, Utrecth, Netherlands, June 6–10 1994. Springer Verlag.

[331] G. M. Wijers and H. E. van Dort. Experience with the use of CASE-tools in the Netherlands. In B. Steinholtz, A. Sølvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference on Advanced Information Systems Engineering (CAiSE'90)*, number 436 in Lecture Notes in Computer Science, pages 5–20, Stockholm, Sweden, May 1990. Springer-Verlag.

[332] G. Wilkie. *Object-Oriented Software Engineering - The Professional Developers's Guide*. Addison-Wesley, 1993.

[333] H. Willars. *Handbok i ABC-metoden (In Swedish)*. Plandata Strategi, 1988.

[334] G. B. Williams, C. K. Mui, B. B. Johnson, and V. Alagappan. Software design issues: A very large information systems perspective. Technical report, CStar, Arthur Andersen, Chicago, 1988.

[335] G. Willumsen. *Executable Conceptual Models in Information Systems Engineering*. PhD thesis, IDT, NTH, Trondheim, Norway, November 1993.

[336] G. Willumsen, J. A. Gulla, O. I. Lindland, and A. H. Seltveit. An integrated environment for validating conceptual models. In Lee et al. [187], pages 353–363.

[337] T. Winograd and F. Flores. *Understanding Computers and Cognition*. Addison-Wesley, 1986.

[338] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[339] L. Wittgenstein. *Philosophical Investigations*. Blackwell, 1958.

[340] M. Yang. *COMIS - A Conceptual Model for Information Systems*. PhD thesis, IDT, NTH, Trondheim, Norway, 1993.

[341] M. Yang, A. H. Seltveit, and A. Sølvberg. A distributed repository management system for formal conceptual IS models. In G. Grosz, editor, *Proceedings of the Sixth Workshop on The Next Generation of CASE Tools*, pages 195–213, Jyväskylä, Finland, 1995.

[342] E. Yourdon. *Managing the System Life Cycle*. Prentice-Hall, 1988.

[343] E. S. K. Yu. An organization modelling framework for multi-perspective information systems design. Technical Report DKBS-TR-93-2, Department of Computer Science, University of Toronto, Canada, 1993.

[344] E. S. K. Yu and J. Mylopoulos. Using goals, rules, and methods to support reasoning in business process reengineering. In Nunamaker and Sprague [238], pages 234–243.

[345] P. Zave. An operational approach to requirements specification for embedded systems. *IEEE Transactions on Software Engineering*, 8(3):250–269, May 1982.

[346] P. Zave. The operational versus the conventional approach to software development. *Communications of the ACM*, 27(2):104–117, February 1984.