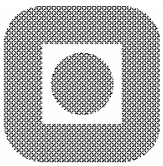
NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

FAKULTET FOR INFORMASJONSTEKNOLOGI, MATEMATIKK OG ELEKTROTEKNIKK



MASTEROPPGAVE

Kandidatens navn:	Magnus Mork
Fag:	Masteroppgave, Datateknikk
Oppgavens tittel (engelsk):	**Building a database backbone for a Natural Language Question Answering System**
Oppgavens tekst:	We now have a running system (GeneTUC) that is able to answer questions about gene and protein interactions. The system is still under development, and the next big step is to make it easier to enter new data into the system. The task will be to make a system that can import existing data from the system, and from users via the web, and when needed, export this data back into the format used by the system (pure text files).

Oppgaven gitt:	01. mars 2004
Besvarelsen leveres innen:	30. juli 2004
Besvarelsen levert:	30 juli 2004
Utført ved:	Institutt for datateknikk og informasjonsvitenskap
Veileder:	Tore Amble og Rune Sætre

Trondheim, 30. juli 2004

Tore Amble
Faglærer

Abstract

The goal of the thesis was to create a database backbone for a Natural Language Question Answering System, and to provide a web interface to this database. The purpose was to make the GeneTUC knowledge base accessible to all participants of the GeneTUC project, and to provide an intuitive and usable interface to the knowledge base.

The GeneTUC system is based on the TUC architecture, and there are several other TUC systems currently in development. A domain independent application was developed to be used with any TUC knowledge base. The application was given the name Ontool. The goal of Ontool was to help improve performance of the GeneTUC system. Therefore, Ontool was designed to be an administrative tool for the knowledge base administrator.

The GeneTUC knowledge base was extended with information from external sources, most notably the Gene Ontology knowledge base. The Ontool application provided support for importing data from OBO- and other various formats.

The Ontool application helped improve the quality of the knowledge base, as it implemented means for preventing duplicates, spelling-errors etc. The knowledgebase also became more accessible as its contents could be accessed in a structural way.

Methods used

Most of the application requirements were unknown and so the project was carried out as a software engineering project with heavy use of prototyping. The plan was to reach the goal of an accessible and usable application through evolutionary prototyping. To assist in the process, the system development theory of Extreme Programming was used.

To help ensure that the usability of the application was good, usability testing with both actual users (biologists and GeneTUC expert), and "normal" users were carried out.

Some initial background research was made to better understand the workflow of the knowledge base administrator, and his interactions with external expert users. The work pattern of the expert users was also studied to better understand their need for a software tool.

Results and contributions

A database backbone was successfully created, and the knowledge base was given a web user interface. Usability tests showed that users were enthusiastic about the application, and found the usability to be good. System tests showed that the application functioned according to the given specifications.

The Gene Ontology knowledge base was imported into the GeneTUC knowledge base. This provided GeneTUC with 17142 new terms and 22304 new relations; resulting in a 1263% increase in knowledge base size.

Preface

This report is the result of the work related to obtaining a Masters degree (Sivilingeniør) in computer science. The work was carried out at the Department of Computer and Information Science, Faculty of Information Technology, Mathematics and Electrical Engineering at the Norwegian University of Science and Technology (NTNU) during the spring semester of 2004.

I would like to thank my supervisors, Rune Sætre and Tore Amble, for valuable help, ideas, and criticism during the project. I would also like to thank Astrid Lægreid, Kristine Misund, and Tone Strømmen at the Medical Research Center (MTFS) at NTNU, for great cooperation and efforts in making the system as good as possible. Their joint contributions have been of great value to the final result.

Additionally, I would like to thank Øyvind Møll at IDI's computer lab, for his quick help and good assistance when publishing the Ontool application.

This report is written in Microsoft Word using font *Palatino Linotype* 12pt.

Cadiz, August 10th, 2004.

_____

Magnus Mork

Contents

List of tables

List of figures

Part I
Introduction

1 Motivation and Background

GeneTUC is a knowledge-based system aimed at extracting knowledge from research articles on molecular biology and genetics, and answer questions given in Natural Language. The system is a part of The Understanding Computer (TUC) project from the Institute of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU), and is built upon the TUC framework put forth by Tore Amble [19].

The TUC framework is a general architecture which, among other things, contains a domain knowledge base. The TUC framework has been applied to a series of systems dealing with natural language and domain knowledge.

The success of a TUC system relies on the quality and quantity of the system knowledge base, as the performance is directly dependent on what the system can infer from its knowledge base. As the TUC architecture is being tested in new domains, the main task is to fill the knowledge base with enough relevant and accurate information. Today, this is a highly manual process with no automatic help or guidance, and with no automatic quality checks. The knowledge base is stored as a set of Prolog predicates, in multiple flat text files. Altering the knowledge base is a slow and painful process, and the system is highly acceptable to errors. This makes it difficult to introduce the system to new users, or to present the knowledge base to other than the core project members.

So far in the TUC project, great work has been done on perfecting and fine-tuning the inner mechanisms of the TUC architecture. However, little has been done to help create and administrate a domain knowledge base that will help TUC systems improve their performance.

Genetics and microbiology is a rich and complex domain, and so GeneTUC's vocabulary is extensive. Research and development efforts in molecular biology and genetics provide new results on a daily basis, and this information must be collected, stored and maintained.

When creating GeneTUC it became apparent that the traditional way of working with the knowledge base was not sufficient. Initiative was taken

by PhD candidate and main driving force in the GeneTUC system, Rune Sætre, to create an automatic system for building, extending, updating and administrating a knowledge base that could be used with TUC systems. The thesis is a direct result of this initiative.

2 Problem Definition and Scope

The below assignment was put forth by IDI's Tore Amble and Rune Sætre.

"**Building a database backbone for a Natural Language Question Answering System**.

We now have a running system (GeneTUC) that is able to answer questions about gene and protein interactions. The system is still under development, and the next big step is to make it easier to enter new data into the system. The task will be to make a system that can import existing data from the system, and from users via the web, and when needed, export this data back into the format used by the system (pure text files)."

The goal is to develop a database web application for administrating the GeneTUC knowledge base. The application should be put to use among the participants of the GeneTUC system. After a trial period, the application can be evaluated, and considered used with other TUC systems. The application to be developed will carry the name Ontool, as it is in fact a tool to be used with (TUC) ontologies.

The thesis is only concerned with the planning, and development of the Ontool application. During development, Ontool will be made available for testing for actual end-users. This, however, is not part of the final evaluation mentioned above, but rather a mean to endure usability of the application. The extensive test-period, and evaluation, is not a topic of the thesis.

3 Project Description

3.1 A comment on the problem definition

A database backbone and a web interface to the knowledge base would hold a number of advantages to the current situation. Shortly put, we have advantages on three levels [DB1]:

1. Moving the knowledge base from its current unstructured format (Prolog predicates scattered around in a set of flat text files), into the structured format of a database.
2. Making the knowledge base accessible for geographically distributed users through the Internet.
3. Presenting the contents of the knowledge base in a highly accessible, easy to use, hyperlink-driven user interface; the World Wide Web (which have proven to be a very usable medium for many other applications).

An incomplete list of advantages would include:

- Better overview of the knowledge base (both the overall status, and the individual details/facts).
- Make the knowledge base more accessible for geographically distributed users.
- Multiple users can update the knowledge base simultaneously.
- Ensure quality in the updating of the knowledge base.
- Easier to import knowledge from other knowledge bases as the target knowledge base is in a structured format.
- Easier to export the knowledge to some other format for presentation, analysis, etc.

3.2 What Ontool is; and is not

The performance of the GeneTUC system relies on its knowledge base. The Ontool application must commit to the structure of the TUC knowledge base architecture. In other words, Ontool is not a tool for extending or altering the TUC knowledge base architecture, but for making sure that the TUC knowledge base architecture is being used to its fullest potential.

The following fictional story can help the reader get an idea of the possibilities and limitations of the Ontool application:

Mr. A is the chief of the GeneTUC project. His goal (in respect to Ontool) is to make GeneTUC understand as much as possible about the gene and protein interactions. When there is something that GeneTUC does not understand, the system can tell this to Mr. A. We can imagine that Mr. A holds a list of words that GeneTUC does not understand. Mr. A wants to find out why GeneTUC does not understand these words and then help GeneTUC learn these words. Using Ontool, Mr. A takes this list and asks which of these words exist in the knowledge base. Ontool presents detailed information about each of the words and their connection to other words (either relations or rules). Mr. A can reach all relevant information, and so discover why GeneTUC did not understand this word by following the hyperlinks presented by the application. Back to the list; Ontool will show which of the words are not known to GeneTUC. Mr. A wants GeneTUC to learn these words, and so he uses Ontool to ask his colleagues down at the center of microbiology for help. He simply uses Ontool to split up the list of unknown words, and sends them to his colleagues. When they log on to the Ontool application they will be shown this list and can start educating GeneTUC. This is done by filling out forms in a step-by-step fashion. Ontool validates the content of the form to keep the user from making mistakes, and checks the knowledge base to avoid duplicates, inconsistencies, errors etc. In doing so, Ontool helps create and maintain a high quality knowledge base which in turn can improve the performance of the GeneTUC system.

Even though the above story only shows *one* use of Ontool, it might help the reader grasp the fundamental concept of Ontool, and shed some light on the application's possible contribution to the future work of TUC systems.

3.3 Constraints and critical factors

The constraints on the thesis are:

- As stated in the Problem definition, the Ontool application will be implemented using a database system, and a web user interface.
- Time is a restrictive factor. The university has set the duration of the master thesis to 20 weeks. During this time the student should show his analytical skills by analyzing the problem domain, his problem solving skills when suggesting and implementing a solution, and his ability to write a report on a technical topic. When dealing with actual end-users, you have to balance the efforts you put into actually implementing the system, and the efforts you do to analyze-, document-, and evaluate it. The users want you to

implement as much as possible, while it is in your best interest to analyze, document, and evaluate to a much greater extent. The objective is to implement as much as possible of the Ontool application and to conduct the usability test with a completed or near-completed application.

- The system should integrate well with other IDI applications/environment.

It is critical that the Ontool application communicates well with the TUC framework. Ontool is a tool to enhance the knowledge base, and GeneTUC has to be able to make use of this knowledge base.

4 Readers Guide

This section provides a brief overview of what can be found in the various chapters of the thesis. The thesis is divided into parts, where each part contains related chapters.

Part I give an introduction to the subject of this thesis. Part II provide a detailed overview of the central aspects needed as a background for the thesis, as well as the research context and an overview of the enabling technologies. Part III describes the process of developing the Ontool application, and testing its usability and functionality. Part IV contains the results of the usability- and system tests, a discussion of the main findings of the thesis and details on future work.

The bibliography is presented towards the end, along with the appendices to the thesis.

Part II
Background and related work

5 GeneTUC

GeneTUC is a knowledge-based system aimed at extracting knowledge from research articles on molecular biology and genetics, and answer questions given in Natural Language. The application is a part of The Understanding Computer (TUC) project from the Institute of Computer and Information Science at the Norwegian University of Science and Technology, and is built upon the TUC framework presented in [19].

Work on GeneTUC was initiated in January 2000 by Anders Andenæs [AA1]. Over the last few years Rune Sætre has been the main driving force in improving the system [17], as he is currently conducting his PhD thesis on the GeneTUC system. For up-to-date information about the system and related research, please refer to Rune Sætre's webpage [18].

In the following, we will present GeneTUC as a knowledge-based system, and inspect its building-blocks. The reason for studying the individual parts of the system is to better understand the architecture- and the fundamental principles which GeneTUC is based on. This knowledge is critical when working to improve the system.

We start by looking at the general framework of GeneTUC (Prolog, TUC). We then move on to the internal structure-, extent-, and the physical storage of GeneTUC's knowledge base (KB). The KB will be the focus of the presentation, because of its direct relevance to the topic of the thesis. We end this chapter by showing the GeneTUC system in action.

5.1 Prolog

Prolog is a simple but powerful programming language developed at the University of Marseille, as a practical tool for programming in logic (PROgramming in LOGic). A logic program is a set of specifications in formal logic, and Prolog uses first-order predicate calculus. Logic, and therefore Prolog, is based on the mathematical notions of relations and logical inference.

Prolog is a declarative language. This means that rather than describing how to compute a solution, a program consists of a data base of facts and

logical relationships (rules) which describe the relationships which hold for the given application. Rather then running a program to obtain a solution, the user asks a question. When asked a question, the run time system searches through the data base of facts and rules to determine, by logical deduction, the answer.

Prolog is commonly used in artificial intelligence (AI) applications such as natural language interfaces, automated reasoning systems and expert systems.

5.1.1 A Prolog program, and its parts

A Prolog program is a set of specifications (facts and rules) in the first-order predicate calculus, describing the objects and relations in a problem domain. The set of specifications is referred to as the database for that problem.

A Prolog program is executed by asking a question (query) about the set of specifications. Queries to the database are patterns in the same logical syntax as the database entries. The Prolog interpreter can therefore use pattern-directed search to find whether these queries logically follow from the contents of the database.

Following is a simple example to show the structure, and execution of a Prolog program.

`-? man(socrates).`
`-? man(X):- mortal(X).`
`-? mortal(socrates).`
`-? Yes`
	fact – Socrates is a man
	rule – all men are mortal
	query – ”Is Socrates mortal?”
	logically derived result – ”Yes”

Table 5.1 Structure and execution of a Prolog program

A ”yes” means that the information in the database is consistent with the subject of the query. If a fact is not deducible from the database, the system replies with a ”no”. This indicated that based on the information available, no such fact is deducible. This is called the *closed world assumption*, and is important in understanding Prolog’s reasoning.

For the purpose of the thesis, the above introduction to Prolog should be sufficient. In chapter 5.3 we will see how the Prolog facts- and rules, and reasoning, is applied to the GeneTUC system.

5.2 TUC

The TUC framework contains Prolog modules for converting natural language into first-order predicates, and vice versa. This makes Prolog capable of reasoning with input given as natural language, and generates natural language as output. The mechanisms of this process are as complex as they are interesting. As this is found to be outside the scope of the thesis, the author would like to recommend [19] as a source of more information. In the following, we will briefly introduce the TUC framework, and show its importance to the GeneTUC system.

Much of the following information has been taken from [19].

TUC (The Understanding Computer) is a name of a project led by Tore Amble, where the aim is to build intelligent knowledge based system with natural language interfaces. When using the word intelligence in the context of TUC, we mean an intelligent agent that communicates in a restricted modality, i.e. natural language text, doing a useful task that requires expertise.

The TUC project was initiated at NTH (Norwegian Institute of Technology, now the Norwegian University of Science and Technolog) in the early 1990's. It was based on a number of previous efforts in creating a natural language interface for querying data bases, among them CHAT-80, PRAT-89 and HSQL. The research goals for the project could be summarized as follows:

- Give computers an operational understanding of natural language
- Build intelligent systems with natural language capabilities
- Study common sense reasoning in natural language

The TUC project seeks to define a language denoted by NRL. This language is as readable as plain English, but has well-defined syntax and semantics. In TUC, NRL serves as both a declarative knowledge definition language, and as a query language.

TUC relies on grammatical analysis for marking sentence elements. A sentence not being grammatically correct (according to TUC's internal grammar), will be rejected without further treatment. ***Enhancing TUC is thus both a question of adding to its vocabulary and semantics, and defining new grammatical constructs.***

The knowledge-based approach, using natural language processing is not without problems. It is important to realize that the knowledge based-

approach relies on semantical, rather than syntactical, analysis of the text.

The preparation of the system, building semantic nets and defining a sensible grammar, is both tedious and time-consuming. Work on the TUC project was started in the early 1990's, but the grammar and semantics are still far from complete.

NOTE: The sentences marked in bold are especially relevant for the work in this thesis. In Part 3 of the thesis we will specify and develop an application for enhancing the GeneTUC knowledge base.

TUC is implemented in Prolog, and so uses Prolog facts- and rules when building a semantic net and defining a sensible grammar. Prolog also drives the reasoning process of TUC.

The TUC framework is domain independent. TUC's design principle is that most of the changes are made in a tabular semantic knowledge base, while there is one general grammar and dictionary. As TUC is based on Prolog, the necessary logic is automatically generated from the semantic knowledge base. When adapting the TUC framework to a new domain, it is necessary to (1) update the grammar to reflect how sound sentences of the domain may be formed, and (2) feed the knowledge base with information to represent the domain. As previously mentioned, this is the main concern of the thesis.

5.2.1 TUC applications

The TUC framework is currently being used in several research projects at NTNU. The author would like to mention the following projects:

- BusTUC. This is an application for answering questions about bus routes in Trondheim.
- LexTUC. This is an application for answering questions about the contents of an encyclopedia (”Store Norske Leksikon”).
- TeleTUC. This is an application for answering questions about the telephone catalogue.
- GeneTUC. This is an application for answering questions about molecular biology and genetics.

The author has purposely avoided details on TUC's inner workings, as this is considered outside the scope of the thesis. There is, however, one

aspect of the TUC framework that needs more looking into; the knowledge base. In the following we will take a look at how the TUC framework utilizes Prolog facts and rules to build a semantic net and defining a sensible grammar.

5.3 Knowledge base

The term knowledge-based system is used to describe programs that reason over extensive knowledge bases, containing facts and rules [11]. The knowledge base is implemented in some formal knowledge representation language; PROLOG and Lisp being the most common.

In the beginning of this chapter we classified GeneTUC as a knowledge-based system in the domain of molecular biology and genetics. We then introduced Prolog as a formal knowledge representation language, and the TUC framework as a mean to translate natural language text to and from a formal language. In the following we will go into details on the facts- and rules that make up the TUC knowledge base, as this is the core of the thesis. Examples are drawn from the GeneTUC domain, when available.

The author will use the notation GeneTUC KB, or simply KB, to refer to the knowledge base of the GeneTUC system.

To allow for a cleaner presentation, the author has chosen to divide the GeneTUC KB in two distinct parts; (1) the semantic network, and (2) the grammar. A justification of this approach, along with a definition of the terminology, will follow in chapter 10.

5.3.1 Semantic network

Finding and organizing information in a systematic way in knowledge management can be an overwhelming challenge. In the field of knowledge systems, the theory of a semantic network is especially important [11].

A semantic network is a knowledge representation scheme based on cognitive psychology models of human associative memory [16]. It is a graph structure which is used to represent associations and relations between objects in a domain. The nodes in the graph are concepts, and the arcs are relations or associations between concepts. The arcs are generally directed and labeled, making a semantic network a directed graph [1].

The term semantic network encompassed a family of graph-based representations. These representations differ chiefly in the names that are allowed for nodes- and links, and the inferences that may be performed on these structures. In the following we will take a look at the semantic network implemented in the GeneTUC system.

The semantic network of GeneTUC

As mentioned earlier, the semantic network is constructed on the bases of nodes and arcs; where the nodes represent the concepts, and the arcs are the relationships between the concepts.

The nodes in the semantic network are nouns relevant to the domain. In the field of molecular biology and genetics the number of possible nouns is astronomical as new nouns are added on a daily basis. The below table shows examples of nodes in the GeneTUC semantic network:

```
Cell, DNA, Gene, Molecule, Organism, Protein, RNA
```

Table 5.2 Example nodes in the GeneTUC semantic network

The arcs in the semantic network are relationships between concepts. The below table shows the types of relationships defined in GeneTUC:

Relation Type	Description
A kind of	Denotes generalisation/specialisation `gene a kind of cellular component`
Is a	Denotes instantiation `chordin is a gene`
Has a	Denotes association `gene has a intron`
A part of	Denotes grouping `lipid_raft a part of membrane`

Table 5.3 Example relations in the GeneTUC semantic network

A word on terminology

The author will use the name *Ontology* (with capital O) to refer to the semantic network of the GeneTUC system. In [12], an ontology is defined as *"a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain, and relations among them*".

5.3.2 Grammar

While it is the work of the Ontology to organize the domain concepts, it is the responsibility of the Grammar to make sense of it all.

The TUC framework relies on semantical analysis of written texts, and it is therefore desirable to connect the concepts to the grammar of natural language. This is done by the part of the KB called the Grammar. The Grammar is actually a set of semantical restrictions, helping the semantical processing of the text. The below table shows the types of semantical restrictions defined in the current GeneTUC Grammar:

Rule Type	Example
adjname_templ	`adjname_templ (calcium, mobilization).`
adjnoun_templ	`adjnoun_templ (antigen, receptor).`
adj_templ	`adj_templ (myocardial, illness).`
adv_templ	`adv_templ (abnormally, large).`
align1	`align1 (cckbr_antagonist, protein).`
attributable	`attributable (asian, asia, continent, city).`
a_compl	`a_compl (responsive, thing, in, cell).`
bm_templ	`bm_templ (be, difficult).`
comp_templ	`comp_templ (eq, animate, object, similar).`
dtv_templ	`dtv_templ (form, with, complex, protein).`
has_class	`has_class (person, father, person).`
iv_templ	`iv_templ (progress, activity).`
measureclass	`measureclass (gram).`
n_compl	`n_compl (of, modulator, transcription).`
ordinal	`ordinal (first, 1).`
particle	`n_compl (of, atherogenicity, particle).`
rv_templ	`rv_templ (demonstrate, activity).`
stanprep	`stanprep (and, thing).`
testclass	`testclass (number).`
tv_templ	`tv_templ (produce, dose, matter).`
v_compl	`v_compl (dimerize, protein, with, protein).`

Table 5.4 Types of semantical restrictions in the GeneTUC Grammar

For an argument on the completeness of the grammar, please refer to [5].

5.3.3 Physical storage of the KB

The GeneTUC knowledge base is currently stored in a set of flat text files. When GeneTUC is executed, the content of these files are read into the Prolog parser and becomes the database of facts- and rules used in the GeneTUC system.

According to [29], the limitations of the file based approach are often:

- Separation and isolation of data
- Duplication of data
- Data dependence
- Incompatible file formats
- Fixed queries/proliferation of application programs

All the above limitations of the file-based approach can be attributed to two factors:
The definition of the data is embedded in the application programs, rather than being stored separately and independently;
There is no control over the access and manipulation of data beyond that imposed by the application programs.

Factor 1 is not valid in the case of GeneTUC, as the KB acts both as data *and* application. Factor 2, however, poses a serious problem to the GeneTUC system. To overcome this problem, it is the opinion of the author that a new approach to KB storage is required. This will be the focus of the thesis.

5.4 Running GeneTUC

In this chapter we have presented the architecture- and principles that the GeneTUC system is built upon. We end by showing a simple example of how the system can be used to answer questions about molecular biology and genetics.

The GeneTUC system is available on the "~busstuc" user on IDI's computers. If the reader needs access to the system, he should contact Tore Amble at IDI, as he is in charge of distributing the password.

1) E: what blocks gastrin.	Question
..... [Which (A)::(gastrin isa substance, (block)/A/gastrin/B, event/real/B)]	Converting natural language into a formal language, and reasons with this.
L740093.	Answer
2) E: what is cck.	Question
..... [which (cck):: cck isa gene]	Converting natural language into a formal language, and reasons with this.
gene	Answer

This simple example shows how GeneTUC (1) accepts natural language as input, (2) converts natural language to a formal language, and (3) answers a question by reasoning on a database of facts and rules.

6 Gene Ontology

The Gene Ontology (GO) project is a collaborative effort to address the need for consistent descriptions of gene products in different databases [6]. The project began as a collaboration between three model organism databases in 1998. Since then, the GO project has grown to include many databases, including several of the world's major repositories for plant, animal and microbial genomes. GO is one of the controlled vocabularies of the Open Biological Ontologies [13].

The GO collaborators are developing three ontologies that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. The below table briefly describes the tree ontologies. Interested readers should refer to appendix E for more information.

Ontology	Description
Molecular Function	the tasks performed by individual gene products; examples are carbohydrate binding and ATPase activity
Biological Process	broad biological goals, such as mitosis or purine metabolism, that are accomplished by ordered assemblies of molecular functions
Cellular Component	subcellular structures, locations, and macromolecular complexes; examples include nucleus, telomere, and origin recognition complex

Table 6.1 The GO ontologies

The author will use the name *Gene Ontology knowledge base* to refer to the set of ontologies shown in the above table.

The Gene Ontology knowledge base is freely available on the Gene Ontology web page [6]. The datasets exist in three different formats: flat files (updated daily), XML (updated monthly) and MySQL (updated monthly).

There are two types of flat file formats, (1) the older GO flat file format and (2) the newer OBO flat file format [13]. The GO flat file format will

continue to be provided alongside the new format, but as the OBO format is more accessible for human reading, and represents the future of GO research, only the characteristics of the OBO format will be described here.

6.1 OBO File format

The format is basically a tag-value format. An OBO document would be structured as follows:

```
<header>
<stanza>
<stanza>
...
```

A "stanza" is a labeled section of the document, indicating that an object of a particular type is being described. Stanzas are structured as follows:

```
[<Object type>]
<tag name>: <tag value>
<tag name>: <tag value>
...
```

For a complete specification of the OBO format, please refer to [14]. In the following, we will present an extract of the Gene Ontology knowledge base file (gene_ontology.obo), and give an explanation of some of the key tags.

```
format-version: 1.0
date: 28:07:2004 16:46
saved-by: jlomax
default-namespace: gene_ontology

[Term]
id: GO:0000001
name: mitochondrion inheritance
namespace: process
def: "The distribution of mitochondria\, including the
mitochondrial genome\, into daughter cells after mitosis or
meiosis\, mediated by interactions between mitochondria and the
cytoskeleton." [PMID:11389764, PMID:10873824, SGD:mcc]
is_a: GO:0048308
is_a: GO:0048311

[Term]
id: GO:0000005
name: ribosomal chaperone activity
namespace: function
def: "OBSOLETE. Assists in the correct assembly of ribosomes or
ribosomal subunits in vivo\, but is not a component of the
assembled ribosome when performing its normal biological
function." [GO:jl, PMID:12150913]
comment: This term was made obsolete because it refers to a class
of gene products rather than a molecular function. To update
```

```
annotations\, consider the molecular function term 'unfolded
protein binding ; GO\:0051082' and the biological process term
'ribosome biogenesis and assembly ; GO\:0042254' and its
children.
is_obsolete: true


[Term]
id: GO:0000019
name: regulation of mitotic recombination
namespace: process
def: "Any process that modulates the frequency\, rate or extent
of DNA recombination during mitosis." [GO:curators]
is_a: GO:0000018
relationship: part_of GO:0006312
...
```

Tag	Description
id	The unique id of the current term.
name	The term name. Any term may only have ONE name defined
namespace	The namespace in which the term belongs.
def	The definition of the current term.
comment	A comment for this term.
synonym	This tag gives a synonym for the term
is_a	This tag describes a sub classing relationship between one term and another..
relationship	This tag describes a typed relationship between this term and another term. The value of this tag should be the relationship type id, and then the id of the target term.
is_obsolete	This tag indicates whether or not the term is obsolete.

Table 6.2 Sample OBO tags and their description

6.2 Gene Ontology and GeneTUC

The Gene Ontology knowledge base consists of some 17637 terms (July 26, 2004) and about 25 000 relations. Compared to the current GeneTUC KB the Gene Ontology knowledge base is enormous.

The GeneTUC group wishes to incorporate the Gene Ontology knowledge base into the GeneTUC KB, in an effort to boost the performance of the GeneTUC system. The thesis is concerned with this project, as the system to be developed should allow for importation of Gene Ontology OBO files.

In regard to the plan of importing OBO contents, the author would like to make some comments on differences and similarities of the GeneTUC- and Gene Ontology knowledge bases. The below table is not an exhaustive list, but serves as interesting reading.

Similarities
Both GeneTUC and GO are heterarchies, as they allow for a term to have multiple parents. The GO ”isa” tag is equivalent to the GeneTUC ”ako” relation The ”part of” relation is defined in both GO and GeneTUC.

Table 6.3 Some similarities between GeneTUC and GO

Differences
GeneTUC require every term to be linked to the top node (thing) trough a (series of) ako relation(s). GO does not require such linking. GeneTUC does not concern itself with information about terms. The relationships between term-names are the focus of the KB, and no other information is stored. GO is much more focused on the term and stores much information about its meaning and significance.

Table 6.4 Some differences between GeneTUC and GO

The requirements aspect of importing information from the Gene Ontology knowledge base will be considered in chapter 11.

7 Enabling Technologies

7.1 The Web

The World Wide Web (hereby referred to only as Web) provides a simple ”point and click” means of exploring the immense volume of pages of information residing on the Internet. Information on the Web is presented on Web pages, which appear as a collection of text, graphics, pictures, sounds, and video, In addition, a Web page can contain hyperlinks to other Web pages, which allow users to navigate in a non-sequential way through information [23].

Much of the Web’s success is due to the simplicity with which it allows users to provide, use, and refer to information distributed geographically around the world. Furthermore, it provides users with the ability to browse multimedia documents independently of the computer hardware being used.

The Web consists of a network of computers that can act in two roles; as servers, providing information; and as clients, usually referred to as browsers, requesting information.

7.1.1 Web servers

Web servers are the computers that actually run web sites. The term "web server" also refers to the piece of software that runs on those computers, accepting HTTP connections from web browsers and delivering web pages and other files (scripts, programs, multimedia-files) to them, as well as processing form submissions. The most common [27] web server software is Apache, followed by Microsoft Internet Information server. Many other web server programs also exist.

7.1.2 Web browser

A web browser is software that helps you navigate through the web. It communicates with web servers via the HTTP protocol, translates HTML pages and image data into a nicely formatted on-screen display, and

present this information to you. Web browsers exist for various types of devices; your personal computer, PDA, cellular phones etc.

The most common web browser [3], by a large margin, is Microsoft Internet Explorer, followed by the open-source Mozilla browser and its derivatives, including Netscape 6.0 and later. Apple's new Safari browser is gaining popularity on Macintoshes running MacOS X, and the Opera shareware browser has loyal followers. The Lynx browser is the most frequently used text-only browser and has been adapted to serve the needs of the vision-impaired.

7.1.3 HyperText Transfer Protocol (HTTP)

The HyperText Transfer Protocol (HHTP) defines how clients and servers communicate. HTTP is a generic object-oriented, stateless protocol to transmit information between servers and clients.

HTTP is based on a request-response paradigm. An HTTP transaction consists of the following stages:

- Connection – The client establishes a connection with the Web server.
- Request – The client sends a request message to the Web server.
- Response – The Web server sends a response (for example, a HTML document) to the client.
- Close – The connection is closed by the Web Server.

HTTP is currently a stateless protocol – the server retains no information between requests. Thus, a Web server has no memory of previous requests. This means that the information a user enters on one page (through a form, for example) is not automatically available on the next page requested, unless the Web server takes steps to make this happen. For most applications, this stateless property of HTTP is a benefit that permits clients and servers to be written with simple logic and run with no extra memory or disk space taken up with information from old requests.

7.1.4 HyperText Markup Language

The HyperText Markup Language (HTML) is a system for tagging a document so that it can be published on the Web. It is a simple, yet powerful, platform-independent document language. HTML has evolved since its introduction in 1992 and is today available in version 4.01. (This is the recommended version from World Wide Consortium, W3C).

HTML has been developed with the intention that various types of devices should be able to use information on the Web: PCs with graphical displays of varying resolution and color depths, cellular phones, hand-held devices, and so on.

Dynamic HTML (DHTML)

Dynamic HTML, or DHTML is not really a new specification of HTML, but rather a new way of looking at, and controlling the standard HTML codes and commands.

When thinking of dynamic HTML, the reader needs to remember the qualities of standard HTML, especially the fact that once a page is loaded from the server, it will not change until another request comes to the server. Dynamic HTML provides means for more control over the HTML elements and allows them to change at any time, without returning to the Web server.

There are four parts to DHTML:

- Document Object Model (DOM)
- Scripts
- Cascading Style Sheets (CSS)
- HTML

Cascading Style Sheets (CSS)

Cascading Style Sheets [24] is a relatively new standard made to complement HTML. Web pages are written in HTML, and the W3C has proposed to separate layout- and contents in a more strict way. They wish to use HTML to only describe the information (the contents), so that the web browsers can show the document in whichever way fits the browser the best. This is in line with the goal of using HTML to make the Web accessible on a variety of devices, and to people with different preferences/handicaps.

W3C has worked with CSS since 1994, and is now developing version 3 of the standard. Most desktop web browsers available today supports version 1, and to some extent version 2 [4]. Other programs have implemented the appropriate profile for their platform: cell phones, PDA, television, printer, speech synthesizer, etc.

Features and benefits of CSS

CSS gives the author (and the user) the possibility to specify how the HTML document should be presented. This can be done internally in the HTML file, or in an external CSS file. The benefits of this are many. One of the best arguments for using CSS is to ease the work of the author. With CSS it is no longer necessary to constantly write FONT FACE, FONT COLOR, BGCOLOR etc, when developing a web page. In stead, the author can keep all the layout-information in a few CSS files, and by editing these files the author can change the layout of multiple HTML documents. This in turn, makes it easier to write, read, and maintain the web pages.

When using CSS, the HTML documents becomes smaller, and this reduces the time needed to download the web page.

In terms of giving the web site a nice and efficient design, CSS is superior to HTML. CSS extends the possibilities, and provides full control when positioning page-elements (either their fixed position, or their position relative to other elements), text formatting, use of color, the framing properties of the elements etc.

CSS is also meant to benefit the user of the web page. The user can specify a stylesheet which is suiting to his needs (ex: a vision-impaired user can specify that all text is shown in double size), and this will work side-by-side with the author's style (in our example; all text will be shown in double size, while all the other styles of the author are kept). This illustrates the cascading property of the CSS standard.

An important benefit of CSS is that it *degrades gracefully*. This means that browsers that do not support CSS will still be able to show web pages designed by CSS. Since the web pages are written in common HTML, all old or text-based browsers will be able to show the contents of the web site. As a consequence, CSS can be used without risk of rendering web pages un-viewable for users with old browsers, or non-graphical browsers like Lynx.

Drawbacks of CSS

Implementation issues, as pages formatted in CSS display differently in various browsers.

Document Object Model (DOM)

The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated [25].

With the DOM, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the DOM.

As a W3C specification, one important objective for the DOM is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM can be used with any programming language.

JavaScript and VBScript

JavaScript and VBScript are web scripting languages from Netscape and Microsoft, respectively. Both languages are interpreted directly from the source code and permit scripting within a HTML document. The scripts may be executed within the browser or at the server before the document is sent to the browser. The constructs are the same, except the server side has additional functionality, for example, for database connectivity.

JavaScript is an object-based scripting language that has its roots in a joint development program between Netscape and Sun, and the syntax resembles Java.

VBScript is a procedural language and so uses subroutines as the basic unit. VBScript grew out of Visual Basic, a programming language that has been around for several years.

Both JavaScript and VBScript are very simple programming languages that allows HTML pages to include functions and scripts that can recognize and respond to user events such as mouse clicks, user input and page navigation. These scripts can help implement complex Web page behavior with a relatively small amount of programming effort.

Cookies

Cookies are a general mechanism which server side connections can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications.

A cookie is a piece of information that the client stores on behalf of the server. The information that is stored in the cookie comes from the server as part of the server's response to an HTTP request. Each time the client visit this server, the browser pack the cookie with the HTTP request. The web server can then use the information in the cookie to identify the user, and depending on the nature of the information collected, possibly personalize the appearance of the Web page. The web server can also add or change the information within the cookie before returning it.

All cookies have an expiration date. If a cookie's expiration date is explicitly set to some time in the future, the browser will automatically save the cookie on the client's hard drive. Cookies that do not have an explicit expiration date are deleted from the computer's memory then the browser closes.

Cookies can be used to store registration information (username/password), preferences (user profile, "My CNN") etc.

Not all browsers support cookies, and some browsers can prevent some or all sites from storing cookies on the local hard drive.

7.2 Database - MySQL

MySQL is a highly capable free relational client/server database system. It is sufficiently secure and stable for many applications, but is commonly viewed as a little brother of the commercial database systems like Oracle, Microsoft SQL Server etc.

In the following, we will present some of the possibilities and limitations of MySQL [10] as this will be relevant in the forthcoming implementation of the Ontool application.

7.2.1 Features of MySQL

Relational Database System

Like almost all other database systems on the market, MySQL is a relational database system.

Client/Server architecture

MySQL is a client/server system. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicates with the server. The clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).

Almost all of the familiar large database systems (Oracle, Microsoft SQL Server, etc) are client/server systems. These are in contrast to the file-server systems, which include Microsoft Access, dBase etc. The decisive drawback to file-server systems is that when run over a network they become extremely inefficient as the number of users grows.

SQL

MySQL supports Structures Query Language (SQL) as its database language. SQL is a standardized language for querying and updating data and for the administration of a database.

There are several SQL dialects available. MySQL follows the ANSI-SQL/92 standard, although with some significant restrictions and extensions.

Programming Languages

There exists a variety of Application Programming Interfaces (API) and libraries for the development of MySQL applications. For client programming you can use, among other, the languages C, C++, Java, Perl, PHP, Python, and TCL.

ODBC

There is an ODBC interface for MySQL. With it MySQL can be addressed by all the usual programming languages running under Microsoft Windows (Delphi, Visual Basic etc).

Platform Independence

It is not only client applications that can run under various operating systems. The MySQL server can also be run under a variety of operating systems. The most significant are Apple Macintosh OS X, IBM OS/2, Linux, Microsoft Windows, as well as countless flavors of UNIX.

Speed

MySQL is considered a fast database system.

7.2.2 Limitations of MySQL

Sub SELECTS

MySQL is not capable of executing a query of the form `SELECT * FROM table1 WHERE x IN (SELECT y FROM table2)`. This limitation can be circumvented in many cases by setting up a temporary table, which, however, is neither elegant nor particularly efficient. In other cases the limitation must be attacked with additional code in an external programming language.

Foreign Keys

MySQL is conversant with foreign keys, which help to link two tables. Usually, however, the keyword foreign-keys also describe the ability of a database to ensure the referential integrity of the linked tables, and at present MySQL is incapable of this. Therefore, in MySQL database applications the programmer must ensure that the integrity of the data

(that is, the relationships among the various tables) is maintained when commands for changing and deleting data are executed.

Transactions

A transaction in the context of a database system refers to the execution of several database operations as a block; that is, as if it was a single command. The database system ensures that either all of the operations are properly executed or else none of them. Thus, for example, it cannot happen that 100 000 NOK is withdrawn from your bank account without then being deposited into mine if an error intervenes (power interruption, computer crash etc). MySQL has some support for transactions, but this feature will take a while to mature and become stable.

Views

MySQL does not support views.

Stored Procedure

MySQL does not support stored procedures.

The lack of Sub SELECTS, Foreign Keys, and Transactions etc does not greatly restrict the possibilities open to client programmers. It does, however, lead to a situation in which the program logic is transferred from the server to the client level. The result is more complex or expensive client programming that would otherwise be the case, leading to redundancies in code, problems in code, and problems with maintenance and alteration of code.

7.3 Integrating Web and Databases - PHP

Hypertext Preprocessor (PHP) is a popular open source HTML-embedded scripting language that is supported by many Web servers including Apache HTTP Server and Microsoft's Internet Information Server, and is the preferred Linux Web scripting language. The development of PHP has been influenced by a number of other languages such as Pearl, C, Java and even to some extent Active Server Pages (ASP), and it supports untyped variables to make development easier. The goal of the language is to allow web developers to write dynamically-generated pages quickly. One of the advantages of PHP is its extensibility, and a number of extension modules have been provided to support such thins as database connectivity, mail and XML.

7.4 Choice of technology

A popular choice when developing small- to medium size web applications, is to use the open source combinations of the Apache HTTP Server, PHP, and the MySQL database system. This is also the technology chosen for the thesis project. A short justification is given below.

As stated under *Constraints* in the *Problem Description* the developed application should be run on IDI's Nova2 UNIX server. Given these constraints, the argument for the chosen technology is as follows:

The chosen technology is widely used, and has a well documented track-record. All applications necessary are free when used in a non-commercial setting, and can run on the desired Nova2 server. On a personal level, the author was curious about both PHP and MySQL, since he had limited experience with the use of these technologies.

One of the goals of the Ontool application was to be a highly usable interface to the GeneTUC KB. DHTML will be used to optimize the user experience.

7.4.1 Pitfalls to avoid

When developing a web-application most of the server-side environment can be controlled and adapted to best suit the web-application. The client-side, however, is a different topic. The diversity in available client-side hardware and software makes it close to impossible to control the client-side environment, and suit it to the application. Indeed, the application

must be adjusted to the environment and not the other way around. Following is a list of potential pitfalls one must have in mind when developing a web-application.

Web server access:

The web server must be configured in such a way as to handle the expected traffic.

Database access:

The database server must be configured in such a way as to handle the expected traffic.

Connection bandwidth:

The amount of unnecessary traffic between server and client should be minimized to avoid a slow system.

Portability:

Care should be taken when creating hyperlinks, so that moving the web-application to a new location is easy.

GUI rendering:

Different web browsers display HTML pages differently. This is a result of proprietary solutions, and failure to agree on standards. This can be a fatal problem to web applications, if not taken seriously. Also, the diversity in available hardware (such as screen size, color resolutions etc) results in different layout of the web-page.

Accessibility:

Some non-standardized HTML tags can create difficulties in parsing the HTML document, and in doing so, create an accessibility problem.

Security:

Web-applications are in general prone to security-attacks. This should be taken into consideration when developing the system.

8 Extreme Programming

Extreme Programming (XP) is a software development methodology, based on the work of Kent Beck, Ward Cunningham, Ron Jeffries, and others [28].

"*Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation*".
- Ron Jeffries, 11/08/2001 [28].

The goal of XP is to help develop high-quality software that provides the highest value for the customer, in the fastest way possible. To do this, XP is based on 12 well-known software development best-practices.

8.1 Best practices of Extreme Programming

Planning Game:

XP planning addresses two key questions in software development: predicting what will be accomplished by the due date, and determining what to do next. The emphasis is on steering the project, rather than on exact prediction of what will be needed and how long it will take. There are two key planning steps in XP, addressing these two questions:

Release Planning is a practice where the Customer presents the desired features to the programmers (through user stories), and the programmers estimate their difficulty. With the costs estimates in hand, and with knowledge of the importance of the features, the Customer lays out a plan for the project. Initial release plans are necessarily imprecise: neither the priorities nor the estimates are truly known, and until the team begins to work, we won't know just how fast they will go.

Iteration Planning is the practice whereby the team is given direction every couple of weeks. During Iteration Planning, the Customer presents the features desired for the next weeks. The programmers break them down into tasks, and estimate their cost.

On-site Customer:

The XP team has continuous access to a real live customer. That is, someone who will actually be using the system. For commercial software with lots of customers, a customer proxy is used instead.

Small Releases:

The team releases running, tested software, delivering business value chosen by the Customer. The most important aspect is that the software is visible, and given to the customer, at the end of every iteration. This keeps everything open and tangible.

Simple Design:

An XP team keeps the design exactly suited for the current functionality of the system. The requirements will change "tomorrow", so the team only does what is needed to meet today's requirements.

Pair Programming:

All production software in XP is built by two programmers, sitting side by side, at the same machine. This practice ensures that all production code is reviewed by at least one other programmer, and is found to result in better design, better testing, and better code.

Test-driven Development:

XP is obsessed with feedback, and in software development, good feedback requires good testing. Top XP teams practice "test-driven development", working in very short cycles of adding a test, then making it work.

Tests in XP come in two basic flavors:

1. Unit Tests are automated tests written by the developers to test functionality as they write it. Each unit test typically tests only a single class, or a small cluster of classes.
2. Acceptance Tests (also known as Functional Tests) are specified by the customer to test that the overall system is functioning as specified. Acceptance tests typically test the entire system, or some large chunk of it.

Design Improvement/Refactoring:

The refactoring process focuses on removal of duplication, and on increasing the "cohesion" of the code, while lowering the "coupling". High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least thirty years. The result is that XP teams start with a good, simple design, and always have a good, simple design for the software. This lets them sustain their development speed, and in fact generally increase speed as the project goes forward.

Continuous Integration:

All changes are integrated into the code base at least daily. The tests have to run 100% both before and after integration.

Collective Code Ownership:

On an XP project, any pair of programmers can improve any code at any time. This means that all code gets the benefit of many people's attention, which increases code quality and reduces defects.

Coding Standard:

XP teams follow a common coding standard, so that all the code in the system looks as if it was written by a single individual. The specifics of the standard are not important: what is important is that all the code looks familiar.

Metaphor:

XP teams develop a common vision of how the program works, which we call the "metaphor". The metaphor use a common system of names to be sure that everyone understands how the system works and where to look to find the functionality you're looking for, or to find the right place to put the functionality you're about to add.

Sustainable Pace:

XP teams are in it for the long term. They work hard, and at a pace that can be sustained indefinitely. This means that they work overtime when it is effective, and that they normally work in such a way as to maximize productivity week in and week out.

8.2 Evaluating Extreme Programming

Extreme Programming is best suited on small- to medium-size projects with total staff of less than 10 people, and total duration of 1-6 months. Because XP is based on the evolutionary prototyping approach it is well suited for *in-house* IT development projects. It is a reasonable choice when the main risks of a project are changing requirements, significant mismatch between project scope and available schedule, and technical staff members that are not currently using advanced software practices.

XP is not well suited to large projects, long projects, or projects with high reliability requirements. It is not well suited to projects that face risks in areas in other than requirements change, schedule, or staff inexperience in advanced software practices.

Main benefits

XP introduces a structured software development methodology to project teams that have previously been exposed primarily to ”code-and-fix” approaches. It provides a training ground that exposes teams to the benefit of more structured software development approaches.

Main risks

Active management is needed to ensure that Extreme Programming does not evolve into ”code and fix”.

8.2.1 Why Extreme Programming in this project

Already in the first talks the author had with his project supervisor, it became clear that this project would benefit from an evolutionary prototyping approach. At this time, the author had only heard about the theory of Extreme Programming, and had no experience with the use of it. It was, however, in the interest of the author to apply some sort of software development theory to the project, so to avoid the ”code and fix” approach.

After some initial research on XP, the author’s curiosity was awoken, and XP was soon the preferred approach. The below table lists some of the reasons for choosing XP:

▪ XP introduces a structured approach to the software development process ▪ The project resources are within the limits of a typical XP project. (time and people) ▪ Users wants to see prototypes ▪ Changing requirements were expected ▪ Ontool has characteristics that comply with typical XP projects

Table 8.1 Why Extreme Programming in this project

9 Usability and Usability Testing

Usability is, according to ISO 13407 "…the effectiveness, efficiency and satisfaction with which specified user can achieve specified goals in particular environments…". Effectiveness is defined as the accuracy and completeness which specified users can achieve specified goals in particular environments. Efficiency is described as the resources expended in relation to the accuracy and completeness of goals achieved. Satisfaction is described as the comfort and acceptability of the work system to its users and other people affected by its use [20].

In layman terms these definitions translates to *do a task as quickly as possible, without using any unnecessary resources and in a way which feels natural to the user*. It might sound easy, but it requires thorough studies to map the user's working pattern, and great skills to apply this pattern to the application.

Usability also depends on the layout of the information when presented on the screen. If the user quickly finds the necessary information he can do a task faster than if he needs to search for it. The optimal situation is when the user does not need any specific information to perform the tasks.

The usability of an application is highly dependent on the user's knowledge of the domain, and previous exposure to the application. An expert needs little or no information to perform a specific task, while a novice needs more time and support to compensate for his limited domain knowledge- and system exposure.

9.1 Usability Testing

Usability testing is a product test approach used to assess the usability of an interface, system or product.

Usability testing can be performed at any level of the product's design. The product may still be in the paper stage, at an early prototype or near completion. Usability tests range from product testing with rigorous controls to more informal exploratory studies. The type of test to use

depends on the objective of the product test, along with the available resources.

There are a number of different types of usability tests, but the most common are exploratory, assessment, validation/verification and comparison tests. The below table presents a short description of these tests. Interested readers should consult [UsabiliyTests] for more information.

Test	**Description**
Exploratory	This test is commonly used in an early stage to assess the efficiency of preliminary design features and concepts - the higher level of the design. The point is to get user feedback on the primary functions of the product. The test participants are asked to suggest improvements on the confusing areas. The designer's task is try to understand why the test users perform the way they do.
Assessment	This is perhaps the most used test type because of its simplicity and straightforwardness. The test is performed after some of the higher level product issues have been revealed by exploratory tests and incorporated into the product. The assessment tests evaluate the lower level operations and aspects of the product based on the findings in the exploratory tests. The tests then examine how effectively the concepts from the exploratory testing have been implemented into the product. The test participants are asked to complete actual tasks and the focus is on information gathering. This test can be both qualitative and quantitative.
Validation/ Verification	This is an objective certification of the product's usability. The focus is on how well the products usability compares to standardized usability performance measures. This test is usually performed before the product is ready for marked release. There is a greater focus on experimental and quantitative information, than in assessment testing, as the product must pass some

	predetermined standard. The test can also be used to evaluate how the product features work together. This test should be performed as early as possible to allow time for the design team to modify the features that did not perform well. This is a quantitative test.
Comparison	This test can be performed at any level of product design in connection with the above mentioned tests. The purpose is to compare two or more design alternatives in terms of which is more usable. It involves collection of both performance and preference data, and the test participants are asked to perform similar tasks with the design alternatives.

Table 9.1 Common usability tests

9.1.1 Discount Usability Engineering-method.

The cost of extensive usability testing can be considerable. Many projects lack the resources to carry out such tests, and choose to disregard usability engineering all together. This is highly unfortunate.

The famous usability guru, Jakob Nielsen [7], was motivated by this when he presented Discount Usability Engineering. He wanted to develop a method which was cheap- and easy to carry out, and still improved the GUI design. His method may not be statistically significant, as it is only based on a few test participants, but has proven to improve the quality of the GUI decisions substantially.

The "discount usability engineering"- method is based on the following three techniques:

- Scenarios
- Simplifies thinking aloud
- Heuristic evaluation

The test participants will go through a realistic scenario and describe *what* they think, and *how* they think, by talking aloud. They will evaluate each dialogue they encounter by using 10 usability heuristics [8]. Although the test is not perfect, it is successfully used to uncover usability problems.

Heuristic evaluation of web applications

Heuristic evaluation is well-suited for the Web because it can be done easy, relatively fast and inexpensive.

Basically, heuristic evaluation involves identifying heuristics, gathering opinions about the usability of the web site, merging and rating the problems that were identified, and then trying to work toward solutions [26].

In the following we will present the 10 usability heuristics of Jakob Nielsen [8], along with a comment on how they apply to web applications [26].

1. Visibility of system status

"The system should always keep users informed about what is going on, through appropriate feedback within reasonable time."

Probably the two most important things that users need to know in a web application, are "Where am I?" and "Where can I go next?" Each page should be branded and indicate which section it belongs to. Links to other pages should be clearly marked. Since users could be jumping to any part of the site from somewhere else, status is needed on every page.

2. Match between system and the real world

"The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order."

As the web application is directed at domain experts, the *user's language* would be very formal and domain oriented.

3. User control and freedom

"Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo."

Many of the *emergency exits* are provided by the browser, but there is still plenty of room to support user control. A *home* button on every page is a simple way to let users feel in control of the application.

4. Consistency and standards

"Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions."

Within the application use wording and buttons consistently. One of the most common cases of inconsistent wording is with links, page titles and site headers.

"Platform conventions" in web applications mean following HTML and other specifications. Deviations from the standards will be opportunities for unusable features to surprise the user.

5. Error prevention

"Even better than good error messages is a careful design which prevents a problem from occurring in the first place."

Because of the limitations of HTML forms, inputting information in web applications is a common source of errors for users. Use of JavaScript can prevent some errors before users submit, but the application should still double-check after submission.

6. Recognition rather than recall

"Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate."

For web applications, this heuristic is closely related to system status. If users can recognize where they are by looking at the current page, without having to recall their path from the home page, they are less likely to get lost.

7. Flexibility and efficiency of use

"Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions."

Some of the best accelerators are provided by the browser; like bookmarks. Important pages of the application should be easy to bookmark. Frames must not be used in a way that prevents users from

bookmarking effectively. If using GET instead of POST on the forms, the users can bookmark the results of a search. When they come back, they get their query re-evaluated without having to type anything in again.

8. Aesthetic and minimalist design

"Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility."

Extraneous information on a page is a distraction and a slow-down. Rarely needed information should be accessible via a link so that the details are there when needed, but do not interfere much with the more relevant content.

9. Help users recognize, diagnose, and recover from errors

"Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution."

Errors will happen, despite all efforts to prevent them. Every error message should offer a solution (or a link to a solution) on the error page.

10. Help and documentation

"Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large."

If the application performs any complicated tasks, the need for help will soon arise. In web application, the key is to not just put up some help pages, but to integrate the documentation into the application. There should be links from the main sections into specific help and vice versa. Help could even be fully integrated into each page so that users never feel like assistance is too far away.

Part III
Developing the Ontool application

10 Problem Analyses

In this chapter we will take a closer look at the problem domain by describing the current situation. A conceptual model of the domain problem will be presented, together with a description of its entities and user profile.

10.1 Overview of the existing system

The GeneTUC Knowledge base, and its structure, was presented in chapter 5.3. Current work on the KB is carried out by manual editing on a set of flat text files. This is done in a normal text editor. The below table list the most common tasks when updating the KB. The terminology used in the table will be explained in chapter 10.1.3.

▪ search for terms and arguments ▪ add, edit, delete or enable/disable a relation X ▪ add, edit, delete or enable/disable a rule Y ▪ editing prolog code

Table 10.1 Common tasks when updating the GeneTUC knowledge base

The text files follows a loose structure, where information is grouped together and stored in alphabetical order. Relation/Rule meta-data, like the name of the author and the date of entry, was sometimes stored to help improve the quality of the KB. Over time, however, the structure of the file had been cluttered, and so a logical *group* of information had become *groups* of information, scattered around in the files.

A text files should only be edited by one person at a time; otherwise information added by one user can be canceled out by another user. There is no mechanism to ensure that this does not happen. However, as the structure of the text files appears chaotic by untrained eyes, Rune Sætre (hereby referred to as *KB administrator* or simply *administrator*) takes care of most of the updates.

There are several people involved in the GeneTUC project. Especially important, in the context of the KB, are the biologists helping the KB administrator to classify and save information about molecular biology

and genetics. When in need of assistance, the administrator sends them a text file of terms which need to be classified. They perform the classifications, save it, and send the file back to the administrator. He then translates this information into Prolog syntax, possibly adds meta-data about the classification, author, entry time etc, and inserts the information into the appropriate KB text file. It is the joint opinion of all participants that this is a tedious, time-consuming and error-prone process.

10.1.1 User profile

The author differentiates the *user* into three roles; (1) the Administrator, (2) the Expert and (3) the GeneTUC User. There is not necessarily a one-to-one relationship between a person and a role. The Administrator can also be an Expert and so on.

The Expert is an expert in the field of linguistics, molecular biology or computer science. He has been granted access to the system by the Administrator, in order to contribute to the GeneTUC KB.

The Administrator is responsible for the development and maintenance of the KB. The administrator controls who has access to the KB, and initiates and overlooks the work on the KB.

The GeneTUC User has questions about molecular biology or genetics, and consults GeneTUC for help. This user role will be overlooked in the rest of the thesis, as we are only concerned with the GeneTUC KB, and not the GeneTUC system as a whole.

The Administrator, Expert and GeneTUC User all access the system using Internet and their personal computer with a web-browser.

10.1.2 Conceptual Model

The conceptual model in the figure below describes the concepts in the problem domain, and the relationships between them. Following the figure is a brief description of the concepts.

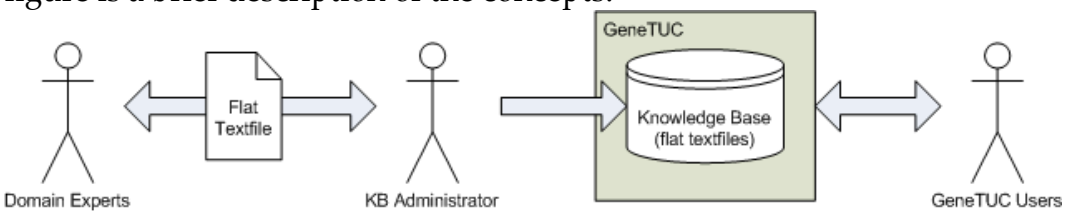


Figure 10.1 Conceptual model of the problem domain

The Expert, Administrator and GeneTUC User have already been described.

GeneTUC

This system consists of a knowledge base. GeneTUC reasons with the contents of the knowledge base, and communicates with the user of the system.

Knowledge Base

The Expert updates the Knowledge Base by adding, editing and deleting its contents. The current KB is organized as shown in the below figure.

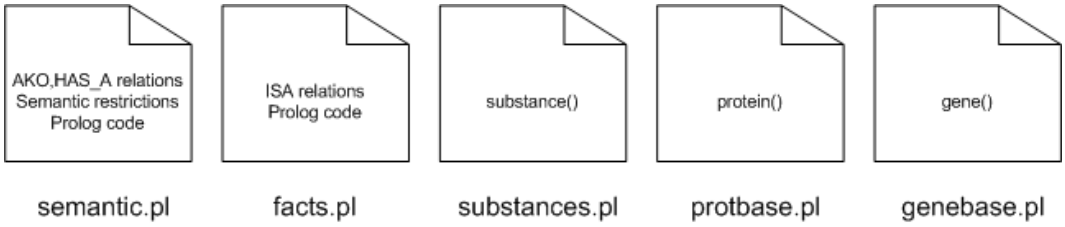


Figure 10.2 Current structure of the GeneTUC KB

10.1.3 Terminology

Throughout the thesis, the author will use a vocabulary suited for the problem domain. The vocabulary is as follows:

Predicate	A grammatical property. A closed set of valid types of grammatical restrictions {a_compl, dtv_templ, …}
Relation type	A closed set of possible relations between concepts (*Terms*) {ako, apo, has_a, isa}
Term	*Terms* are nouns (concepts) which are relevant in the domain. The terms can enter into one or more *Relations* and *Rules* (as arguments)
Argument	All possible words. Can be adjective, adverbs, verbs, nouns etc (*Term* is a subset of *Argument*). The *Argument* is a part of a *Rule*
Relation	Consists of one *Relation type*, and two *Terms*
Rule	Consists of a *Predicate* and an arbitrary number of *Arguments*. The number of arguments depends on the Predicate.

Ontology	The set of all *Terms* and *Relations*
Grammar	The set of all *Rules*
GeneTUC KB	The set of *Ontology* and *Grammar*

Table 10.2 Thesis terminology

To help clarify any confusion, the above information is depicted in the below figure.

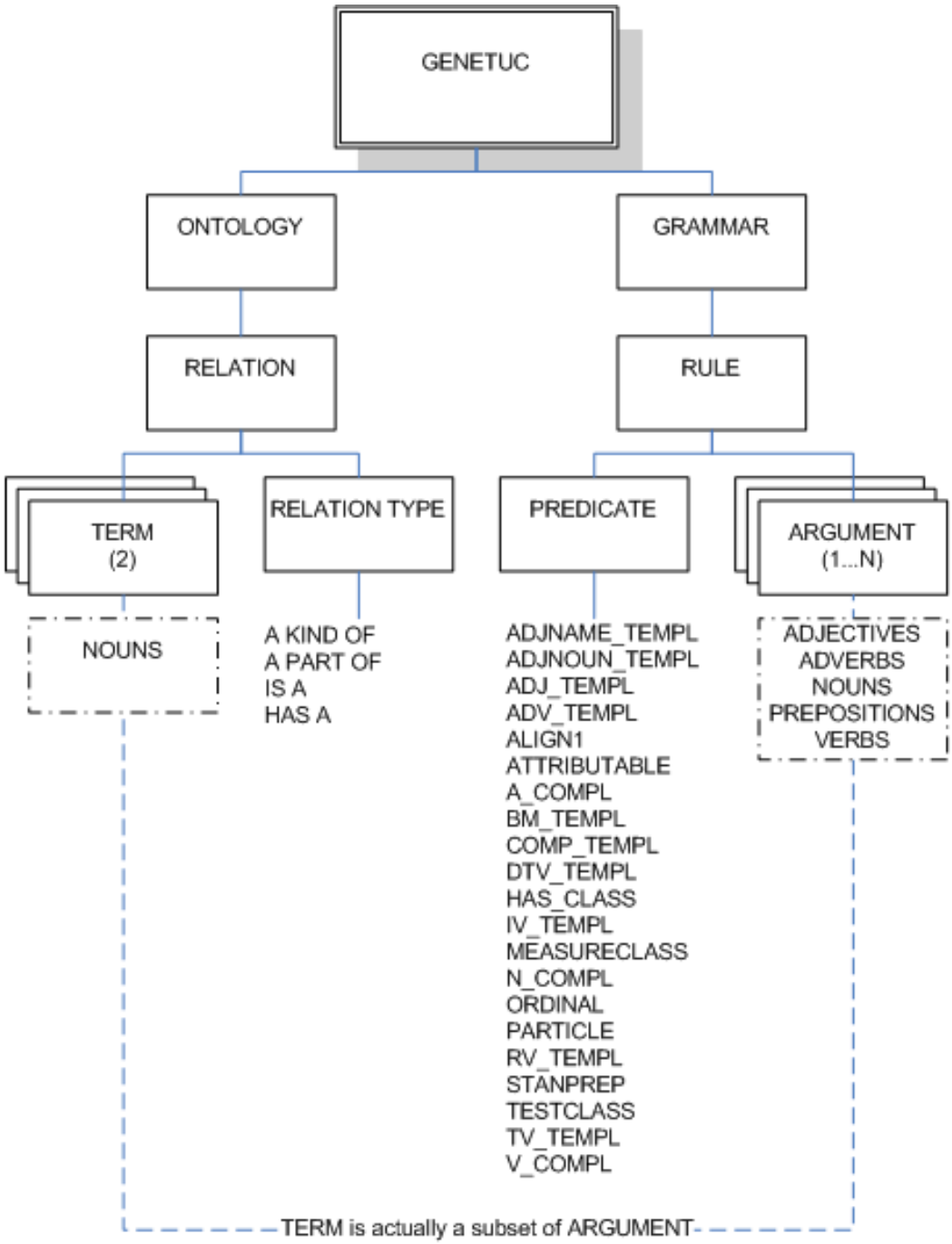


Figure 10.3 Thesis terminology - A graphical representation.

11 Requirements Specification

The functional and non-functional requirements state the functionality and the quality of the Ontool application. The functional requirements are listed in table 11.1, before they are further elaborated by use cases in chapter 11.1.1. The non-functional requirements are listed and described in chapter 11.3. Specifications for the Ontool database is provided at the end of the chapter.

11.1 Functional requirements

The below table presents an overview of the functional requirements for the Ontool application. The requirements are given a unique identity; F-#, where the F stands for Functional. They are described briefly, and prioritized. The priority shows how important a requirement is, and this will be reflected in the implementation. The requirements with a High priority will be implemented first, the requirements with a Medium priority next, and in the end the requirements with a Low priority will be implemented.

Requirement	Description	Priority
F-1	Log on to the application	M
F-2	User Administration	H
F-3	Message Forum	M
F-4	Assign tasks to users.	M
F-5	Show a personalized opening screen, with ”My Assignments”.	M
F-6	Show Ontology as a tree structure.	H
F-7	Show Grammar.	H
F-8	Search Ontology.	H
F-9	Search Grammar.	H
F-10	Add a new relation to the KB.	H
F-11	Edit an existing relation.	H
F-12	Delete a relation.	H
F-13	Enable/Disable a relation.	H
F-14	Add new rule to the KB.	H
F-15	Edit an existing rule.	H
F-16	Delete a rule.	H

F-17	Enable/Disable a rule.	H
F-18	Show a chronological list of updates to the KB	M
F-19	Review Ontology.	M
F-20	View statistics about the KB.	M
F-21	Import flat text files into KB. (also Gene Ontology)	H
F-22	The possibility to manually type in Prolog code that does not interact with the application DB.	M
F-23	Export KB contents to Prolog file format.	H
F-24	The possibility to export KB to other formats.	L
F-25	Help on using the application	M
F-26	Add a new term to the KB.	H
F-27	Edit an existing term.	H
F-28	Delete a term.	H

Table 11.1 Functional requirements of the Ontool application

11.1.1 Use cases for Ontool

A use case captures a user-visible function or an interaction between the system and the actor [22]. In the case of the Ontool application, it is reasonable to divide the actor in two categories; the domain-expert, and the administrator. The domain-expert and administrator will have different use of the application, as shown in the following use cases.

Actor	Domain-Expert
Description	The Expert is an expert in the field of linguistics, biology or computer science. He has been granted access to the application by the Administrator, in order to contribute to the GeneTUC KB.

Actor	Administrator
Description	The Administrator is responsible for the development and maintenance of the KB. The administrator controls who has access to the KB, and initiates and overlooks the work on the KB.

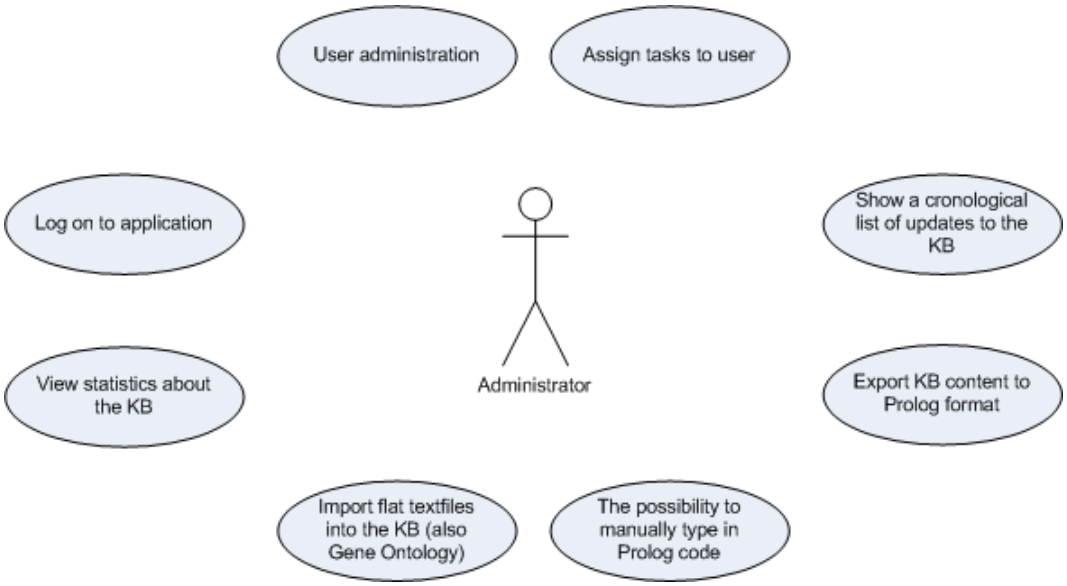


Figure 11.1 Administrator use cases

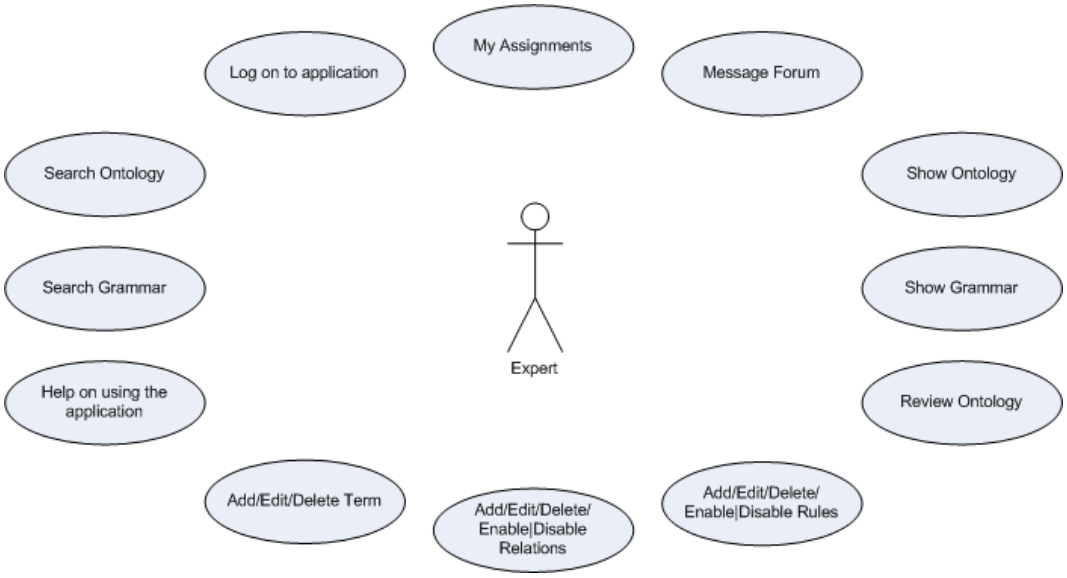


Figure 11.2 Expert use cases

USF-1: "Log on to application"

Actor	Expert, Administrator
Summary	The actor wishes to access the KB.
Precondition	The user has been given a username/password by the administrator, and can connect to the server.
Basic course of events	1. The actor executes the command to "Log on to the application". 2. The actor inputs a username and password. 3. The application checks the validity of the username/password. If the pair is valid, the user is granted access to the system.
Alternative paths	In step 2, the actor can choose to cancel the operation. This terminates the use case.
Exception paths	In step 3, the system might experience the following problems: 1. Problems with connecting to the application can lead to problems verifying the username/password. 2. The actor has given invalid username/password. 3. Both cases should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The actor has full access to the application.

USF-2: "User Administration"

Actor	Administrator
Summary	The actor wishes to administrate users' access to the KB.
Precondition	The actor is logged on.
Basic course of events	1. The actor executes the command to "Administrate user access". 2. The application presents a list of users who currently can access the system. 3. The actor can modify this list, by a. Add a new user b. Edit a user c. Delete a user
Alternative paths	
Exception paths	In step 2 and 3, the system might experience the following problems: 1. Problems with connecting to the KB can lead to problems displaying the list of current users, and modifying the list. This should be explained to the actor. Use case is then terminated.
Postcondition	The administrator can see, and control, who has access to the system.

USF-3: "Message Forum"

Actor	Expert, Administrator
Summary	The actor wishes to write a message to-, or read messages from, other users of the system.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to open the "Message Forum". 2. The application should present the most recent messages posted in the forum (the actor can choose how many messages to show). 3. The actor can post a new message to the forum.
Alternative paths	
Exception paths	In step 2 and 3, the system might experience the following problems: 1. Problems with connecting to the KB can lead to problems with displaying-, or saving- messages to the forum. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The Message Forum is used as a way of communication between users of the system. The users can post comments, or instructions to each other.

USF-4: "Assign tasks to users"

Actor	Administrator
Summary	The actor wishes to assign classification tasks to expert users. When the experts carry out these tasks, the KB is updated and optimized for better performance.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Assign tasks to users". 2. The application presents a list of terms to be classified (orphan terms) 3. The actor selects the terms to be classified from the list, or types it in manually. Then the expert user to carry out the classification is selected. 4. The application saves this information, and notifies the expert user.
Alternative paths	In step 2, the actor can choose to import a list of terms for classification, or the system shows the orphan terms of the KB. In step 3, the actor can choose to select all terms, or divide the list into N equal size portions.
Exception paths	In step 2, the system might experience the following problems: 1. There are no orphan terms in the KB. This should be explained to the actor. 2. Problems with importing the file. This should be explained to the actor, along with the reason for the failure. Use case is then terminated. In step 3, the system might experience the following problems: 1. Problems with connecting to the KB can lead to problems with displaying the name/identity of the expert users. This terminates the use case. In step 4, the system might experience the following problems: 1. Problems with saving the tasks. This terminates the use case.
Postcondition	After the administrator has assigned a task to an expert user, the expert user should be notified. This notification can be done via email, or on the screen when the expert user logs onto the system the next time.

USF-5: "Show a personalized opening screen, with My Assignments"

Actor	Expert
Summary	The actor wishes to see which tasks he has been assigned.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to ”Show My Tasks”. 2. The application presents a list of tasks that has been assigned to the respective expert, by the administrator. 3. Each task is represented with a hyperlink, to start carrying out the task.
Alternative paths	In step 2, if no tasks have been assigned to the user; create a hyperlink to ”Request task from administrator”.
Exception paths	In step 2, the system might experience the following problems: 1. Problems with connecting to the KB can lead to problems with displaying the tasks.
Postcondition	The actor has an overview of tasks to do. He can then start carrying out the tasks (described in other Use Cases)

USF-6: "Show Ontology as a hyperlink tree structure"

Actor	Domain-Expert
Summary	The user wishes to study (the structure of) the Ontology.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to show the ontology as a hyperlink tree structure. 2. The application shows the ontology-root (THING) and its children as a hyperlink tree structure. 3. By clicking on any term, the tree structure will expand or contract accordingly, to show or hide the children of this term accordingly. If a term has no children, this should be clearly stated to the user.
Alternative paths	In step 3, the actor might want to know more about a term, and should be presented with a link to "Show more information about this term".
Exception paths	In step 2, the system might experience the following problems: 1. Problems with connecting to the KB. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated. 2. Ontology-root (term THING) does not exist in KB. The system tells the Actor and terminates the use case.
Postcondition	The user can study the Ontology by simply clicking his way through the hyperlink tree structure.

USF-7: "Show Grammar"

Actor	Domain-Expert
Summary	The user wishes to see the semantic restrictions that apply to the Ontology.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to show the semantic restrictions. 2. The application shows all semantic restrictions, on the form: "predicate(arg1,...,argN)." Meta-data about this predicate should also be presented (when was this predicate added, and who did it?) 3. All arguments are presented as hyperlinks. By clicking an argument, the application will show all semantic restrictions related to that argument. (ex: Clicking the "person" argument in "v_compl(drive, person, car).", will show all restrictions where "person" is given as an argument) 4. If the argument is a valid term, a hyperlink to the Ontology should be provided. 5. Hyperlinks to the following operations should be present: a. "Add a new rule to the Grammar" b. "Edit an existing rule" c. "Delete a rule" d. "Comment away a rule"
Alternative paths	In step 3, after clicking the argument. The actor should be able to go back to the previous screen..
Exception paths	In step 2, the system might experience the following problems: 1. Problems with connecting to the KB. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated. 2. No semantic restrictions exist for the Ontology. The system tells the Actor and terminates the use case.
Postcondition	The user can study the semantic restrictions of the Ontology by simply clicking his way through the hyperlink argument-structure.

USF-8: "Search Ontology"

Actor	Domain-Expert
Summary	The user wishes to search for term names in the Ontology.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to search the Ontology. 2. The application shows a search interface similar to Google/Altavista etc, where the user can enter a search string. 3. The application should present the results of the search in an orderly fashion, and all term names should be hyperlinks to show all present information about the term and its relations.
Alternative paths	In step 2, the user should be able to specify whether he wishes an exact search, or a wider search (equivalent to SQL "=" and "LIKE"). In step 3, when the result is presented, the user should be able to perform another search immediately, without any navigation.
Exception paths	In step 3, the system might experience the following problems: 1. Problems with connecting to the KB. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated. 2. Search returns 0 results. The system tells the Actor, and resumes normal execution.
Postcondition	The user can study the search result, and use the hyperlinks to navigate around in the Ontology.

USF-9: "Search Grammar"

Actor	Domain-Expert
Summary	The user wishes to search for predicates- or arguments in the Grammar, to see which semantical restrictions apply.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to search the Grammar. 2. The application shows a search interface similar to Google/Altavista etc, where the user can enter a search string. 3. The application should present the results of the search in an orderly fashion with meta data. This presentation should be similar to the "Show Grammar" use case.
Alternative paths	In step 3, when the result is presented, the user should be able to perform another search immediately, without any navigation.
Exception paths	In step 2, the system might experience the following problems: 1. Problems with connecting to the KB. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated. 2. Search returns 0 results. The system tells the Actor, and resumes normal execution
Postcondition	The user can study the search result, and use the hyperlinks to navigate around in the Grammar.

USF-10: "Add a new relation to the KB"

Actor	Domain-Expert
Summary	The actor wishes to add a new relation to the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Add a new relation to the KB". 2. The application presents a form for inserting data about the relation, and automatically inserts information about author name, and entry date. If the relation involves a term which has been imported from the Gene Ontology knowledgebase, the application should display the Gene Ontology Graph View for that term. 3. The user fills out the desired fields and submits the form. 4. Before submitting the form, the application checks to see if the required fields are filled out. 5. The application then checks if the terms in the relation exist in the KB. If not, the user is confronted with this information, and is asked for an appropriate course of action. The user can choose to add the undefined term, choose another term from a list of terms with similar spelling, or rewrite the name of the term. 6. Based on the chosen action, the application adds the relation to the KB.
Alternative paths	In step 2, the application should provide a link so that the actor can manually search the Gene Ontology knowledge base. In step 3 and 5, the actor can choose to terminate the use case. In step 4, if not all required fields are filled out, the application informs the actor on which fields are necessary.
Exception paths	In step 2,5 and 6, the system might experience the following problems: 1. Problems with connecting to the KB will result in no automatic filled out fields, no validity checking or adding of the relation.
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-11: "Edit an existing relation"

Actor	Domain-Expert
Summary	The actor wishes to edit an existing relation in the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to ”Edit an existing relation”. 2. The process is the same as the ”Add a new relation to the KB”, except that the form fields are filled out with information from the KB.
Alternative paths	See ”Add a new relation to the KB”.
Exception paths	See ”Add a new relation to the KB”
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-12: "Delete a relation"

Actor	Domain-Expert
Summary	The user wishes to delete a relation from the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to ”Delete a relation”. 2. The application asks the user to confirm the operation. 3. Upon confirmation, the application permanently deletes the relation from the KB. 4. The application should log this activity, to help oversee the development of the KB.
Alternative paths	In step 2, the user can abort the operation. The use case is then terminated.
Exception paths	In step 3, the system might experience the following problems: 1. Problems with connecting to the KB will keep the relation from being deleted. This should be explained to the actor, and the actor might be presented with a link to ”Tell System Administrator about this problem”. Use case is then terminated. In step 4, the system might experience problems with connecting to the file system. This will result in no logging of the activity.
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-13: "Enable/Disable a relation"

Actor	Domain-Expert
Summary	The user wishes to disable a relation from the KB. This means that the relation will not be deleted, but will no longer affect the KB. Relations that are disabled can be enabled again; thus bringing them back into the KB.
Precondition	The user is logged in.
Basic course of events	1. When the application displays relations, the actor should be able to click a hyperlink, or check a checkbox, to enable/disable the relation.
Alternative paths	If a relation is disabled it should be able to enable it, and vice versa.
Exception paths	In step 1, the system might experience the following problems: 1. Problems with updating the KB. This should be explained to the actor, and the actor might be presented with a link to ”Tell System Administrator about this problem”. Use case is then terminated
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-14: "Add a new rule to the KB"

Actor	Domain-Expert
Summary	The actor wishes to add a new relation to the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Add a new rule to the KB". 2. The application presents a form for inserting data about the rule, and automatically inserts information about author name, and entry date. 3. When the user selects the desired predicate for the rule, the application should present fields for typing in arguments. 4. The user fills out the desired fields and submits the form. 5. Before submitting the form, the application checks to see if the required fields are filled out. 6. The application adds the rule to the KB.
Alternative paths	In step 3 and 4, the actor can choose to terminate the use case. In step 6, if an identical rule exist in the KB the actor is notified. This terminates the use case.
Exception paths	In step 2 and 6, the system might experience the following problems: Problems with connecting to the KB will result in no argument fields to fill out, and no adding of the rule.
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-15: "Edit an existing rule"

Actor	Domain-Expert
Summary	The actor wishes to edit an existing rule in the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Edit an existing rule". 2. The process is the same as the "Add a new rule to the KB", except that the form fields are filled out with information from the KB.
Alternative paths	See "Add a new rule to the KB"
Exception paths	See "Add a new rule to the KB"
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-16: "Delete a rule"

Actor	Domain-Expert
Summary	The user wishes to delete a rule from the KB
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Delete a rule". 2. The application asks the user to confirm the operation. 3. Upon confirmation, the application permanently deletes the rule from the KB. 4. The application should log this activity, to help oversee the development of the KB.
Alternative paths	In step 2, the user can abort the operation. The use case is then terminated.
Exception paths	In step 3, the system might experience the following problems: 1. Problems with connecting to the KB will keep the rule from being deleted. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated. In step 4, the system might experience the following problems: 1. Problems with connecting to the file system will result in no logging of the activity.
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-17: "Enable/Disable a rule"

Actor	Domain-Expert
Summary	The user wishes to disable a rule from the KB. This means that the rule will not be deleted, but will no longer affect the KB. Rules that are disabled can be enabled again; thus bringing them back into the KB.
Precondition	The user is logged in.
Basic course of events	When the application displays relations, the actor should be able to click a hyperlink, or check a checkbox, to enable/disable the relation..
Alternative paths	If a rule is disabled it should be able to enable it, and vice versa.
Exception paths	In step 1, the system might experience the following problems: 1. Problems with updating the KB. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The KB has been updated to improve the performance of GeneTUC..

USF-18: "Show a chronological list of updates to the KB"

Actor	Expert
Summary	The actor wishes to see the recent changes in the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Show recent updates to the KB". 2. The actor specifies a date to start the search from, and the application presents a list of all changes in the KB from that date. Other relevant information should also be shown.
Exception paths	In step 2, the system might experience the following problems: Problems with connecting to the KB can lead to problems with displaying the changes to the KB. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The actor can, in an orderly fashion, see the chronological changes to the KB..

USF-19: "Review Ontology"

Actor	Expert
Summary	The actor wishes to review the work of other experts.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Review the work of other experts". 2. The actor specifies which level of confidence/user to review. The application presents a list of classifications, according to the search-specifications.
Alternative paths	
Exception paths	In step 2, the system might experience the following problems: 1. Problems with connecting to the KB can lead to problems with displaying the classifications. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The actor can see which classifications other experts are uncertain of. The actor can then choose to edit these classifications (see Use Case).

USF-20: "View statistics about the KB"

Actor	Expert, Administrator
Summary	The actor wishes to see statistics about the KB.
Precondition	The actor is logged on.
Basic course of events	1. The actor executes the command to "View statistics about the KB". 2. The application presents statistics about: a. Unique terms in the KB b. Relations (ako, apo, has_a) c. Semantic restrictions (v_compl, v_templ, ...)
Exception paths	In step 2, the system might experience the following problems: 1. Problems with connecting to the KB can lead to problems showing the statistics. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The actor can quickly get an overview of the extent of the KB.

USF-21: "Import flat text files into KB (also Gene Ontology)"

Actor	Expert, Administrator
Summary	The actor wishes to import a file into the KB
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to ”Import a file”, then selects the desired filename and format. 2. The application reads the contents of the file, and (not in the case of OBO files) allow the user to preview the parse-result before the data is inserted into the KB
Alternative paths	
Exception paths	In step 2, the system might experience the following problems: 1. Problems with connecting to the KB can lead to problems with importing data.
Postcondition	The actor can, in an orderly fashion, see the chronological changes to the KB..

USF-22: "The possibility to manually type in Prolog code that does not interact with the application DB"

Actor	Domain-Expert
Summary	The user wishes to manually type in prolog predicates, without changing the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Manually input prolog predicates". 2. The application presents a free text area where the user can input prolog code. 3. The application should also present a warning to user, making it clear that the application does not process the contents of the free-text area (the actor is fully responsible for the contents of the free text area). 4. The actor should be able to save the changes to the free-text area, or cancel the operation.
Alternative paths	In step 4, if the user saves his changes, the application will present those changes in the free text area (for manually editing) in the future.
Exception paths	In step 4, the system might experience the following problems: 1. Problems with saving the changes. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The actor can manually type in prolog predicates. This gives the actor complete control over the KB.

USF-23: "Export KB contents to Prolog file format"

Actor	Administrator
Summary	The user wishes to export the contents of the KB, into a format which can be used by Prolog.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Export KB into Prolog file format". 2. The application should create a file, and write the complete contents of the KB to that file. When writing the file, the application should follow a set of guidelines to make it compatible with Prolog, and the TUC applications. 3. The actor should be able to specify the name of the file to be created, along with writing a file header (comments, author, date etc).
Alternative paths	In step 2, the actor should be able to cancel the operation. This ends the use-case.
Exception paths	In step 2, the system might experience the following problems: 1. Problems with creating the file. This should be explained to the actor, and the actor might be presented with a link to "Tell System Administrator about this problem". Use case is then terminated.
Postcondition	The actor can use the newly created file in TUC applications.

USF-24: "The possibility to export KB to other formats"

Actor	Domain-Expert
Summary	The user wishes to be able to export the KB to a variety of formats.
Precondition	The user has access to details on the Ontool database structure.
Basic course of events	1. The actor decides on the required output format, and consults the Ontool database documentation. 2. As the KB is stored in a database format, the actor can write a program or script using his favorite programming language to output information from the KB in the desired format.
Alternative paths	
Exception paths	
Postcondition	The actor can access the KB contents in the desired format.

USF-25: "Help on using the application"

Actor	Domain-Expert
Summary	The user wishes help from the application to carry out a task.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Help on using the application". 2. The application presents a help page, which contains a list of tasks that can be carried out using the system. Each task is given a detailed description of how to proceed to carry out the respective task.
Alternative paths	In step 1, the command to "Help on using the application" can be found on a general menu, or in the page where the task is performed.
Exception paths	In step 2, the system might experience the following problems: 1. No help is provided for the task. Information on how to contact people for more information should be provided. Use case is then terminated.
Postcondition	The actor will hopefully find the help useful, and be able to carry out the desired task.

USF-26: "Add a new term to the KB"

Actor	Domain-Expert
Summary	The actor wishes to add a new term to the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Add a new term to the KB". 2. The application presents a form for inserting data about the term, and automatically inserts information about author name, and entry date. 3. The user fills out the desired fields and submits the form. 4. Before submitting the form, the application checks to see if the required fields are filled out. 5. The application adds the term to the KB.
Alternative paths	In step 3 and 4, the actor can choose to terminate the use case. In step 6, if an identical term exist in the KB the actor is notified. This terminates the use case.
Exception paths	In step 2 and 5, the system might experience the following problems: 1. Problems with connecting to the KB will result in no argument fields to fill out, and no adding of the term.
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-27: "Edit an existing term"

Actor	Domain-Expert
Summary	The actor wishes to edit an existing term in the KB.
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to "Edit an existing term". 2. The process is the same as the "Add a new term to the KB", except that the form fields are filled out with information from the KB.
Alternative paths	See "Add a new term to the KB"
Exception paths	See "Add a new term to the KB"
Postcondition	The KB has been updated to improve the performance of GeneTUC.

USF-28: "Delete a term"

Actor	Domain-Expert
Summary	The user wishes to delete a rule from the KB
Precondition	The user is logged in.
Basic course of events	1. The actor executes the command to ”Delete a term”. 2. The application asks the user to confirm the operation. 3. Upon confirmation, the application permanently deletes the term from the KB, as long as the term is not involved in any relations to other terms. 4. The application should log this activity, to help oversee the development of the KB.
Alternative paths	In step 2, the user can abort the operation. The use case is then terminated.
Exception paths	In step 3, the system might experience the following problems: 1. Problems with connecting to the KB will keep the term from being deleted. This should be explained to the actor, and the actor might be presented with a link to ”Tell System Administrator about this problem”. Use case is then terminated. In step 4, the system might experience the following problems: 1. Problems with connecting to the file system will result in no logging of the activity.
Postcondition	The KB has been updated to improve the performance of GeneTUC.

11.2 System specification for the Ontool database

The database system specification lists important features for the Ontool database.

Initial database size

- The are approximately 1600 terms, involved in about 6600 relations.
- There are about 2700 grammatical rules.

Database rate of growth

- When importing the Gene Ontology knowledge base, the number of terms will rise to about 20 000, and the number of relations will be close to 25 000. The process of adapting the Gene Ontology knowledge base to the GeneTUC KB, is expected to create around 2000 new relations, but few new terms.
- The number of rules will increase with the number or terms added.

The type and average number of record searches

- Searching for relations and terms is expected to be the most frequent, although very modest, with about 100-500 per day.
- Searching for rules; approximately 50-100 per day.

Networking and shared access requirements

- All registered users should be able to connect to the centralized database located at an IDI server. The system should allow for at least 10 users accessing the system simultaneously.

Performance

- Expect less than 1 second response time for all single record searches. (This is related to the web server – database server performance; not to the web-server – end user performance)

Security

- The database should be password-protected.

11.3 Non-functional requirements

A non-functional requirement is a property of a system as it runs, but it can also be a property such as maintainability, portability and extensibility, which are related to the application development. It may not be directly observational as a functional requirement, but nevertheless specifies how the system should act.

The table presents the non-functional requirements of the Ontool application, together with a description of what they mean in this setting.

Requirement	Description
Portability	Hardware independence: ▪ The client-side of the application should be accessible from any normal/modern PC, using any normal/new/common operating system and a normal/new internet browser. ▪ The server-side of the application should be run on one of IDI's servers, (and should work fairly independent.)
Reliability	Accuracy: ▪ The system should have a good precision of computations and output. Error-tolerance: ▪ It is important that the application does not hang if the user submits wrong input.

Table 11.2 Non functional requirements of the Ontool application

12 System Architecture and Design

The system architecture defines building-blocks, their interaction and what functionality each block is responsible for performing. The overall architecture describes the Ontool application in broad terms. The deployment diagram illustrate where the different system components resides. The chapters about the client- and server architecture explain the responsibilities of the components.

12.1 A tired-architecture

Web based applications are distributed applications. They utilizes the resources of multiple machines or at least multiple process spaces, by separating the application functionality into more manageable groups of tasks that can be deployed in a wide variety of configurations. There are a number of benefits to dividing applications into pieces, not the least of which are reusability, scalability, and manageability.

Ultimately, dividing up an application in this manner results in the creation of a series of application layers or tiers, each of which is responsible for an individual element of the applications processing. Distributed applications are often divided into three or more tiers.

The Ontool application is based on a three tier architecture, where the tiers are; (1) User Services tier, (2) Business Services tier, and a (3) Data Services tier.

12.1.1 Component architecture

The below table present the various components in the Ontool architecture.

Component	Description
Web Server	Ontool is relatively web server agnostic. Apache 1.3.29, or newer, is recommended, but Microsoft IIS is also supported. Essentially, if the web server supports PHP, Ontool will work.
PHP	Ontool operates with PHP v4.3.4. The PHP runtime engine parses the Ontool pages and associated functions.
Database	MySQL version 4.0.17, or newer, is supported by Ontool.

Server	Ontool interacts with the database by connecting on a standard port to the configured host and database using a username and password.
GeneTUC database	The database is managed by the database server. It consists of multiple tables. See chapter 12.3 for more information.
Operating System	The operating system is relatively transparent to Ontool with the exception of such platform-specific characteristics as path and file names. Ontool should run on Linux, FreeBSD, Windows, Macintosh OS and other platforms that support PHP and MySQL
File system (server)	While the KB resides in the database, the application pages, templates, and other components that constitute Ontool reside in the native file system. ▪ All Ontool pages reside in the `public_html` directory ▪ All images reside in the `public_html/images` directory ▪ All log-files reside in the `public_html/log` directory ▪ Documentation reside in the `public_html/doc` directory
Client Browser	Ontool uses some DHTML, and this somewhat limits the support for client browsers. Ontool works with Mozilla/Gecko-based browsers as well as Microsoft Internet Explorer and Opera, as long as they are newer than 2000/2001. Older browsers can also be capable of running Ontool, but no guarantee is provided. Ontool utilizes JavaScript. According to [3], 92% of Internet surfers use browsers that support JavaScript. Ontool is designed to be used on a screen with 1024x768 pixels. According to [3] , 57% of the Internet users use this resolution on their screen. Ontool is optimized for Microsoft Internet Explorer version 6, as this the by far most popular web browser among Internet users [3]; 72% use IE6.
Networking	Ontool uses the HTTP protocol for client-server communication. The use of a three-tier architecture allows the information transfer between the web server and the database server to be optimized. The communications between these systems do not have to be based on the

	Internet standards but can use faster, low-level communications protocols.
Security	Cookies will be used to save username/password on the client machines, so that users can be recognized and verified.
Hardware	No definite hardware specifications are given as this is dependent on the chosen platform and software configuration. As long as the hardware is capable of successfully running the above software, it is sufficient to run the Ontool application.

Table 12.1 Components of the Ontool application

12.1.2 Brining the components together

The components described in the above table will be brought together as illustrated in the below figure.

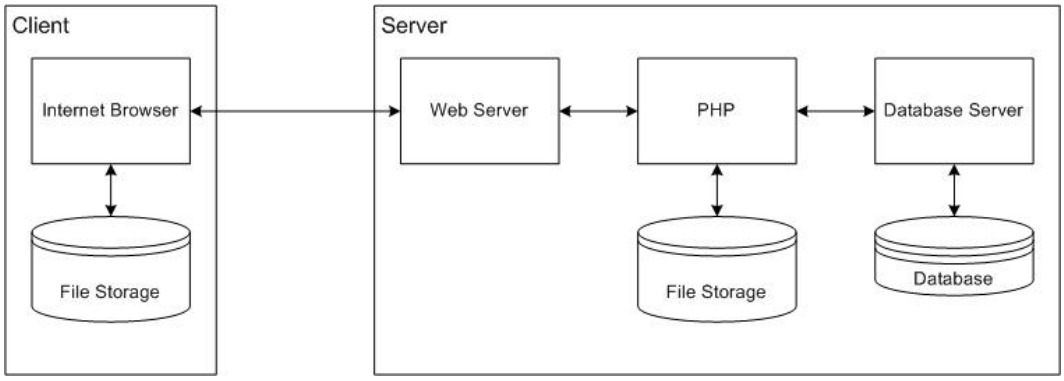


Figure 12.1 The Ontool component architecture

The Ontool application will have components on both the client- and server side. We will discuss this further in the following.

12.1.3 The Server side's responsibilities

The main part of the Ontool application will reside on the serve, as a set of PHP files. When the user requests a certain function of the application, the web server handles the incoming HTTP request and executes the PHP runtime engine to parse the appropriate PHP page. The web server sends all PHP files to the PHP runtime engine, which parses and executes the file. All database access is incorporated into the PHP file, and so it is handled by the PHP MySQL engine. The result of this execution is sent

back to the web server; as a HTML document. This document is then sent to the client, as a respond to the initial HTTP request.

12.1.4 The Client side's responsibilities

The HTML document sent to the client is parsed and the content is presented by the browser. Unfortunately, different browsers interpret and display HTML code slightly different, as mentioned in an earlier chapter.

The HTML documents received by the browser can contain client-side code that is executed by the browser. This client-side code helps create a more dynamic application, capable of adjusting quickly to user actions.

12.1.5 Data Service tier

The Data Layer has the responsibility to provide information to the Business Logic Layer based on data received from the Business Logic Layer, and to update the data bases in the system.

The Ontool application operates on data stored in a database, and the file system. The Data Layer managing this, consists of

- MySQL database server and PHP MySQL Functions library
- The Operating System and PHP File system Functions library

12.1.6 Business Service

In general, the Business Logic components have the following main responsibilities:

- Receive data from the Client Layer
- Send data to the Client Layer
- Send data to the Data Layer
- Receive data from the Data layer
- Control the data flow in the application

The business layer is managed by the PHP files These files are constructed as shown in the below figure.

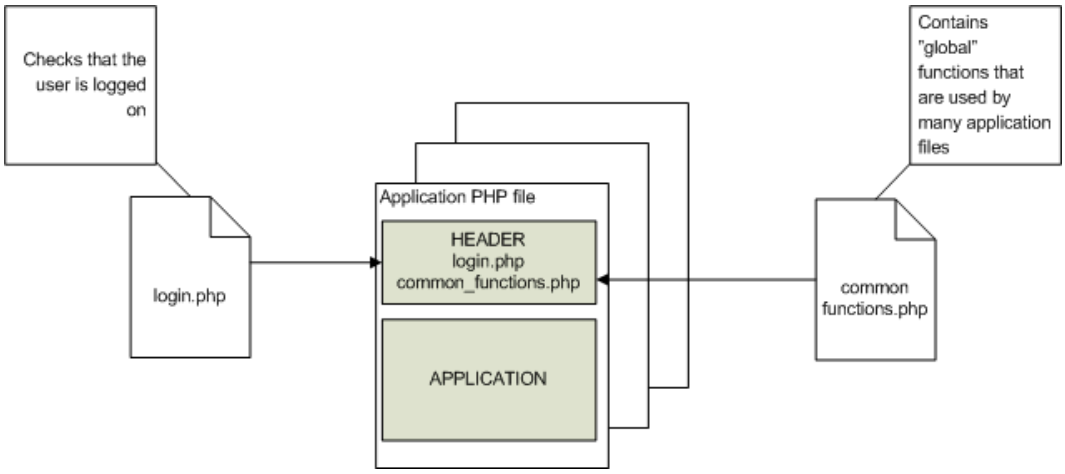


Figure 12.2 Ontool PHP file architecture

12.1.7 User Service

The User Service tier creates a visual gateway for the user to interact with the application. The layer is managed by DHTML (HTML, CSS, DOM, and JavaScript).

12.2 Evaluation of the architecture

This evaluation will shed some light over the choices made in the architecture phase. The decisions were based on the functional- and non-functional requirements, along with the project goals and limitations.

The overall system is a 3-tier architecture. This is a normal way of dividing a system which has a GUI, system logic and data storage. The greatest advantage of this approach is that the presentation, the application processing and the data management are logically separate processes. The disadvantage is the complexity it introduces, compared to a simpler 2-tier architecture.

During development the web- and database server have been located on the same computer. This is not necessarily the case when the system is finally published, and as we will see next; the chosen architecture is portable.

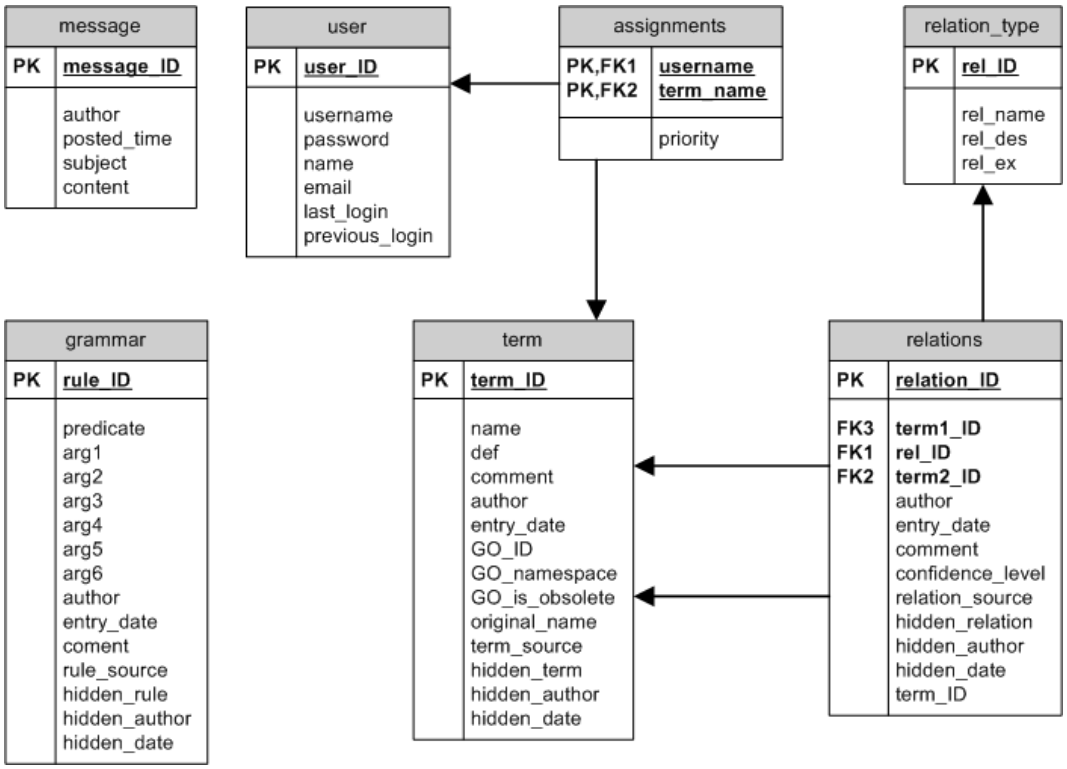
12.2.1 Portability

There are no components in the architecture using any device-specific features so the architecture is portable under the following conditions:

- The Ontool application can be moved to any web-server, as long as the web server supports PHP.
- The database can be moved to any server that supports MySQL
- If the database is moved to another server than the web-server, the `$mysql_host` variable in the database connection function must be set accordingly.

As stated in the Functional Requirements (chapter 11.1), users should access Ontool through their web browsers, and the Ontool system should be situated on IDI's servers. In the architecture described in this chapter, these requirements are fulfilled. For a justification on the choice of technology in the architecture (Apache, PHP, MySQL), please refer to chapter 7.4.

12.3 Database design



12.3.1 Relation Type

Attribute	Description
rel_ID	Unique ID for this relation type
rel_name	The name of the relation type, spelled out in text
rel_des	A description of this type of relation
rel_ex	An example showing the relation type in use

12.3.2 Assignments

Attribute	Description
username	Specifies the user which has been given the assignment
term_name	Specifies the name of the term to be classified
priority	Specifies the level of priority the user should give to classifying this term

12.3.3 User

Attribute	Description
user_ID	Unique ID
username	Specified the system name of the user
password	Specifies the system password for the user
name	Specifies the real name of the user
email	Specifies the email address of the user
last_login	Specifies the date when the user logged on "this time"
previous_login	Specifies the date when the user was last logged on

12.3.4 Message

Attribute	Description
message_ID	Unique ID
author	Specifies who inserted/edited this record
posted_time	Specifies when this record was inserted/edited
subject	Specifies the message header
content	Specifies the message body

12.3.5 Term

Attribute	Description
term_ID	Unique ID
name	Specified the rule type
def	Argument 1 of the rule
GO_ID	The Gene Ontology ID (if existing for this term)
GO_namespace	The Gene Ontology namespace (if existing for this term)
GO_is_obsolete	This tag specifies if the term is obsolete in the Gene Ontology knowledge base
original_name	Holds the name of the term, as it was before the Ontool application enforces rules on term names
author	Name of the user who inserted/edited this record
entry_date	The date when the record was inserted/edited
comment	Any comments the author might have
term_source	The origin of the record (name of file, web, etc)
hidden_rule	Specifies if the record is enabled or disabled
hidden_author	Specifies which user disabled the record
hidden_date	Specifies when the record was disabled

12.3.6 Relations

Attribute	Description
relation_ID	Unique ID
term1_ID	ID of the first term involved in the relation
rel_ID	Specifies the type of the relation
term2_ID	ID of the second term involved in the relation
author	Name of the user who inserted/edited this record
entry_date	The date when the record was inserted/edited
comment	Any comments the author might have
relation_source	The origin of the record (name of file, web, etc)
hidden_relation	Specifies if the record is enabled or disabled
hidden_author	Specifies which user disabled the record
hidden_date	Specifies when the record was disabled

12.3.7 Grammar

Attribute	Description
rule_ID	Unique ID
predicate	Specified the rule type
arg1	Argument 1 of the rule
arg2	Argument 2 of the rule
arg3	Argument 3 of the rule
arg4	Argument 4 of the rule
arg5	Argument 5 of the rule
arg6	Argument 6 of the rule
author	Name of the user who inserted/edited this record
entry_date	The date when the record was inserted/edited
comment	Any comments the author might have
rule_source	The origin of the record (name of file, web, etc)
hidden_rule	Specifies if the record is enabled or disabled
hidden_author	Specifies which user disabled the record
hidden_date	Specifies when the record was disabled

13 Ontool System Test

The purpose of software testing is to assess and evaluate the quality of work performed at each step of the software development process. In other words; to reveal bugs, weaknesses and other errors. Another objective is to ensure that the system is doing what it is actually meant to do.

The Ontool testing process includes

- a set of sub-unit tests
- two system tests
- a quality requirements test

All tests were carried out by the author. All test behavior was thoroughly checked before a test result was set. The tests were carried out while the system resided on the following platform:

Operating system	Microsoft Windows XP
Web server	Apache 1.3.29 (Win32)
PHP	PHP 4.3.4 for Windows
MySQL server	MySQL 4.0.17-nt
Web browser	Microsoft Internet Explorer 6 Opera 7

Table 13.1 Platform used when testing Ontool

The documented tests involve central parts of the Ontool application. An explicit test of every single aspect of the application was not possible due to lack of resources. An indirect test of these aspects was carried out during prototyping and implementation. Each documented test is given a description, a link to functional requirements, and a test-result.

In this chapter we will present the executed tests, and comment on the result of these tests. The test conclusion and discussion is presented in part IV of the thesis.

13.1 Sub-unit tests

The sub-unit tests concentrates on the individual sub-units of the application, to ensure that they have the intended behavior. Test results and conclusions will be further explained in chapter 16.3.

Sub-unit tests (with F-#)	**Description**	**Result**
Logging on (F-1)	The test is passed if a user can log on with a valid username/password. An invalid username/password will not log the user on,	Passed
User administration (F-2)	The test is passed if the user can: ▪ retrieve user information from the database ▪ add a new user ▪ retrieve the new user information ▪ edit user information ▪ retrieve the new user information again ▪ delete a user	Passed
Message Forum (F-3)	The test is passed if a user can: ▪ add a new message to the database ▪ retrieve it from the database ▪ edit and save it again ▪ retrieve it again ▪ delete the message ▪ The user should not be able to edit or delete messages written by others.	Passed
Assign tasks to users (F-4)	The test is passed if a user can: ▪ add a list of terms to a user ▪ retrieve the list from the database ▪ edit the list, and save it again ▪ retrieve the list again ▪ delete the list	Passed
GeneTUC Settings (no F#)	The test is passed if the user can: ▪ Retrieve the settings from the file ▪ Edit the settings, and save them ▪ Retrieve the settings from the file again	Passed
Show a personalized opening page, with ”My Assignments”	The test is passed if the user is presented with a personalized opening page when logging onto the application, and this page contains information and links to help the user carry out his tasks.	Passed

Show Ontology as a hyperlinked tree structure. (F-6)	The test is passed if the user can show the ontology as a hyperlinked tree structure, where it is possible to click a term to show its related terms.	Passed
Show Grammar. (F-7)	The test is passed if the user can show all rules in the KB Grammar, and all arguments are presented as search hyperlinks.	Passed
Search Ontology (F-8)	The test is passed if the user can search for terms in the ontology, and the result is presented as a set of hyperlinks to show all information about the terms.	Passed
Search Grammar (F-9)	The test is passed if the user can search for rules (both predicates and arguments), and the result, among with all meta data is presented to the user.	Passed
Edit terms (F-26,27,28)	The test is passed if a user can: ▪ add a new term to the KB ▪ retrieve the term from the KB ▪ edit and save the term again ▪ retrieve the term again ▪ delete the term	Passed
Edit relations (F-10,11,12,13)	The test is passed if a user can: ▪ add a new relation to the KB ▪ retrieve the relation from the KB ▪ edit and save the relation again ▪ retrieve the relation again ▪ enable and disable the relation ▪ delete the relation	Passed
Edit rules (F-14,15,16,17)	The test is passed if a user can: ▪ add a new rule to the KB ▪ retrieve the rule from the KB ▪ edit and save the rule again ▪ retrieve the rule again ▪ enable and disable the rule ▪ delete the rule	Passed
Show a chronological list of updates to the KB (F-18)	The test is passed if the user can get a list of all TERMS, RELATIONS and RULES with a more recent entry_date than the date specified.	Passed

Review classifications (F-19)	The test is passed if the user can search for classifications, by author and confidence level, and limit the search to a specific number of matches. All classifications in the result should be presented with meta-data.	Passed
View statistics about the KB (F-20)	The test is passed if the user can access statistical information about the contents of the KB.	Passed
Import flat text files into KB. (also Gene Ontology) (F-21)	The test is passed if the user can import rules and relations from flat text files into the KB. The rules and relations should pass an extensive validity check before they are stored in the KB. Rules and relations that already exist in the KB should not be overwritten.	Passed
The possibility to manually type in Prolog code that does not interact with the application DB (F-22)	The test is passed if the user can manually edit Prolog code through the Ontool interface. The Prolog code should not affect the Ontool database, but should be directed at the GeneTUC system. The Prolog code mentioned, should be exported along with the Ontool database when the user chooses to ”export the KB to Prolog file format”.	Passed
Export KB to Prolog file format. (F-23)	The test is passed if the user can export KB contents (rules and relations) to the Prolog file format. The manually typed Prolog code mentioned in F-22, should also be exported.	Passed

Table 13.2 Sub unit tests for the Ontool application

13.2 System test

When specifying the functional requirements of the system, two actors were identified. For this reason, there are presented two system test; one for each type of user (actor). The system test consists of a set of sub-unit tests, and is passed only if all of the sub-systems are passed.

System test # 1	The expert contributes to the contents of the KB
Description / Sub Unit tests	▪ Log on to the application ▪ Message Forum ▪ Show a personalized opening screen, with ”My Assignments” ▪ Show Ontology as a hyperlink tree structure ▪ Search Ontology ▪ Show Grammar ▪ Search Grammar ▪ Edit term ▪ Edit relation ▪ Edit rule ▪ Review Ontology
Result	Passed

Table 13.3 System test #1 for the Ontool application

System test # 2	The administrator maintains the KB
Description / Sub Unit tests	▪ Log on to the application ▪ User Administration ▪ Assign tasks to users ▪ Show a chronological list of updates to the KB ▪ View statistics about the KB ▪ Import file ▪ The possibility to manually type in Prolog code that does not interact with the application DB. ▪ Export KB to Prolog file format
Result	Passed

Table 13.4 System test #2 for the Ontool application

13.3 Non-Functional Requirement Tests

Quality requirement	Description
Performance	Start-up: The time needed from the user start the application, until the moment the application is ready to be used. ▪ The test is passed if the application is ready within 5 seconds, 10 out of 10 times. Command execution: The time from the user executes a command, to the functionality is ready to be used. ▪ The test is passed if the application is ready within 5 seconds, 10 out of 10 times.
Portability	The system should be able to run on different platforms, as long as the all necessary software is present. The test is to deploy the application to two different platforms, and execute the system test (described earlier) Platform 1: Microsoft Windows Operating system, with IIS web server, PHP and MySQL for Windows database server. Platform 2: Linux Operating system, Apache web server, PHP and MySQL database server.
Robustness/ Reliability	The application must be able to run even if the user submits wrong input, i.e. letters when expecting numbers and vice versa. ▪ The test is passed if the system does not crash, and notifies the user about wrong input.

Table 13.5 Non-functional requirement test for the Ontool application

The non-functional requirement tests were carried out as part of the sub-unit and system tests. The results and conclusion on all system tests will be discussed in chapter 16.3.

14 Ontool Usability Test

As part of the development process, the author performed two usability tests on the Ontool application. This chapter is concerned with the planning and execution of these tests. Results of the tests will also be shown. The conclusion and discussion of the two tests will be presented in Part IV of the thesis.

14.1 Usability test # 1

This test was carried out in the end of May when Ontool was still in an early prototype stage. The Ontology part of the application is concerned with searching, presenting, and updating the Ontology (terms and relations). As this part was relatively stable and functional, it was the topic for the first usability test.

14.1.1 Test participants

The author wanted to test the prototype on the actual end users of the system, as this feedback would be highly interesting for the further development of the prototype. The test participants were selected from the group of potential end users of the Ontool application. From these, 3 biologists and 1 expert on the GeneTUC system was selected. All participants had extensive domain knowledge, and knowledge of ontologies. All participants were used to using computers and had experience from surfing the web.

To complement the feedback from actual users, the author decided to test the application on participants with no domain knowledge, or knowledge of ontologies. These users were selected from the university student body (NTNU). None of these participants had ever worked with ontologies, but they were all experienced with using computers and the web.

The table on the next page summarizes the participants in the usability test # 1.

Category	Actual end user	Average university ”Joe”
Domain knowledge (Genetics)	X	
Knowledge about Ontologies	X	
Experience with Web	X	X
Sum participants	4	4

Table 14.1 Participants in usability test #1

14.1.2 Test

Due to lack of resources, the author could not be present when all users performed the test. Therefore, a ”do it yourself” test was designed.

The test consisted of a list of tasks (scenarios) which the user would perform. Upon finishing a scenario, the user was asked to answer a question relevant to that task. This would act as a control to see if the scenario was completed successfully or not. The user was also asked to state the time used in performing the scenario. This would be used to measure the efficiency with which the user could carry out the tasks.

A pilot test was carried out with one of the expert users. During this test the author was present, and could follow the action- and performance of the user. As no major flaws were uncovered in the pilot test, the test went ahead as planned.

When testing the *Average (university) Joe*, the author gave a brief introduction to ontologies and the structure of the GeneTUC Ontology. The expert users were given no such additional information.

The author met with the participants after the execution of the test, and both the test and its initial results were discussed.

14.1.3 Results

The author received a completed questionnaire from 6 of the users. The below figure presents the average of the 6 users, rounding to closest whole number. The conclusion of the usability test # 1 will be discussed in chapter 16.4.

Questionnaire

		1	2	3	4	5	6	7	
Computer experience									
I believe my knowledge about computer systems is	BAD	○	○	○	◉	○	○	○	GOOD
Computer System Usability Questionnaire		1	2	3	4	5	6	7	
Overall, I am satisfied with how easy it is to use system	DISAGREE	○	○	○	○	◉	○	○	AGREE
I am able to complete my work quickly using system	DISAGREE	○	○	○	○	◉	○	○	AGREE
I am able to efficiently complete my work using system	DISAGREE	○	○	○	○	◉	○	○	AGREE
I feel comfortable using system	DISAGREE	○	○	○	○	◉	○	○	AGREE
It was easy to learn to use system	DISAGREE	○	○	○	○	○	○	◉	AGREE
I believe I became productive quickly using system	DISAGREE	○	○	○	○	○	◉	○	AGREE
System gives me error messages that clearly tell me how to fix problems	DISAGREE	○	○	○	○	○	◉	○	AGREE
Whenever I make a mistake using system, I recover easily and quickly	DISAGREE	○	○	○	○	○	◉	○	AGREE
The information (such as online help, on-screen messages, and other documentation) provided with system is clear	DISAGREE	○	○	○	◉	○	○	○	AGREE
It is easy to find the information I needed	DISAGREE	○	○	○	○	◉	○	○	AGREE
The information provided for system is easy to understand	DISAGREE	○	○	○	○	◉	○	○	AGREE
The information is effective in helping me complete the tasks	DISAGREE	○	○	○	○	○	◉	○	AGREE
The interface of system is pleasant	DISAGREE	○	○	○	○	○	○	◉	AGREE
I like using the interface of system	DISAGREE	○	○	○	○	○	◉	○	AGREE
System has all the functions and capabilities I expect it to have	DISAGREE	○	○	○	○	◉	○	○	AGREE
Overall, I am satisfied with system	DISAGREE	○	○	○	○	◉	○	○	AGREE
Screen and perceptual limitation		1	2	3	4	5	6	7	
Reading characters on the screen	HARD	○	○	○	○	◉	○	○	EASY
Organization of information	CONFUSING	○	○	○	○	◉	○	○	VERY CLEAR
Sequence of screens	CONFUSING	○	○	○	○	◉	○	○	VERY CLEAR
Does it provide easily distinguised colours	BAD	○	○	○	○	○	◉	○	GOOD
Are menues distinct from other displayed information?	BAD	○	○	○	○	○	◉	○	GOOD
Are input data fields visually distinct from other displayed information?	BAD	○	○	○	○	○	◉	○	GOOD
Is the screen density reasonable?	BAD	○	○	○	○	○	◉	○	GOOD
Terminology and system information		1	2	3	4	5	6	7	
Use of terms throughout system	INCONSISTENT	○	○	○	○	◉	○	○	CONSISTENT
Terminology related to task	NEVER	○	○	○	○	◉	○	○	ALWAYS
Position of messages on screen	INCONSISTENT	○	○	○	○	○	◉	○	CONSISTENT
Prompts for input	CONFUSING	○	○	○	○	○	◉	○	CLEAR
Computer informs about its progress	NEVER	○	○	○	○	◉	○	○	ALWAYS

Figure 14.1 Results of usability test # 1

Positive aspects commented upon

- Easy to use
- Lucid interface
- Good error messages

Negative aspects commented upon

- Too many open windows
- Missing functionality

14.2 Usability test # 2

This test was carried out in the middle of July when work on the Ontool application was almost completed. The application, as it will be delivered, was stable and rich on functionality for both the Ontology- and the Grammar part of the GeneTUC KB.

The author was preparing a rather extensive usability test which would function as (1) a test of the delivered system (see System Test), (2) a measure of the applications usability, and (3) a bases for suggestions on future work on the application.

Due to limited time and resources in the final stage of the project, the usability test # 2 became somewhat crippled. The execution- and results of the test will be provided in the following.

14.2.1 Test participants

The original plan for usability test # 2 called for use of the same participants as in usability test # 1. Due to geographical distance, and problems with the availability of the Ontool application, the plans for usability test # 2 was revised. The below table summarizes the test participants.

Category	Average university ”Joe”
Domain knowledge (Genetics)	
Knowledge about Ontologies and use of Grammar to restrict the semantic of an ontology.	
Experience with Web	X
Sum participants	3

Table 14.2 Participants of usability test # 2

We note that the expert users are omitted in this test. This will affect the conclusions which can be drawn from the test result. The author also wishes to inform that only one of the participants in usability test # 2 is the same as in usability test # 1.

14.2.2 Test

In usability test # 2 the author was present during test execution. This allowed for direct monitoring-, and verbal communication with the participant while executing the test. The test consisted of a list of tasks to perform (scenario), and the participants were asked to evaluate a selection of screen dialogs in respect to a chosen set of heuristics. Following these principles, the usability test # 2 was actually based on the ”discount usability engineering” method presented in chapter 9.1.1. The heuristics used were alos presented in this chapter.

As mentioned earlier, the usability test # 2 would also function as a system test, and so the scenario was similar to the system tests (S1 and S2) presented in chapter 13. To ease the evaluation of the usability test, the scenario was shortened somewhat, and the focus would lie on only a few key dialogs that the author considered problematic.

The author started the test by introducing the participants to the concept of ontologies, and explained the structure of the GeneTUC KB, focusing on the Ontology and Grammar. The participants then carried out the test, one by one, with only the author present in the room.

14.2.3 Results

The following pages show extracts of the GUI that the participants evaluated. For each task in a scenario, the participants evaluated the dialogues according to 10 heuristics. The author has calculated the average score for each heuristic, and presents the results along with the GUI extracts.

The conclusion of the usability test # 2 will be discussed in chapter 16.4.

User administration

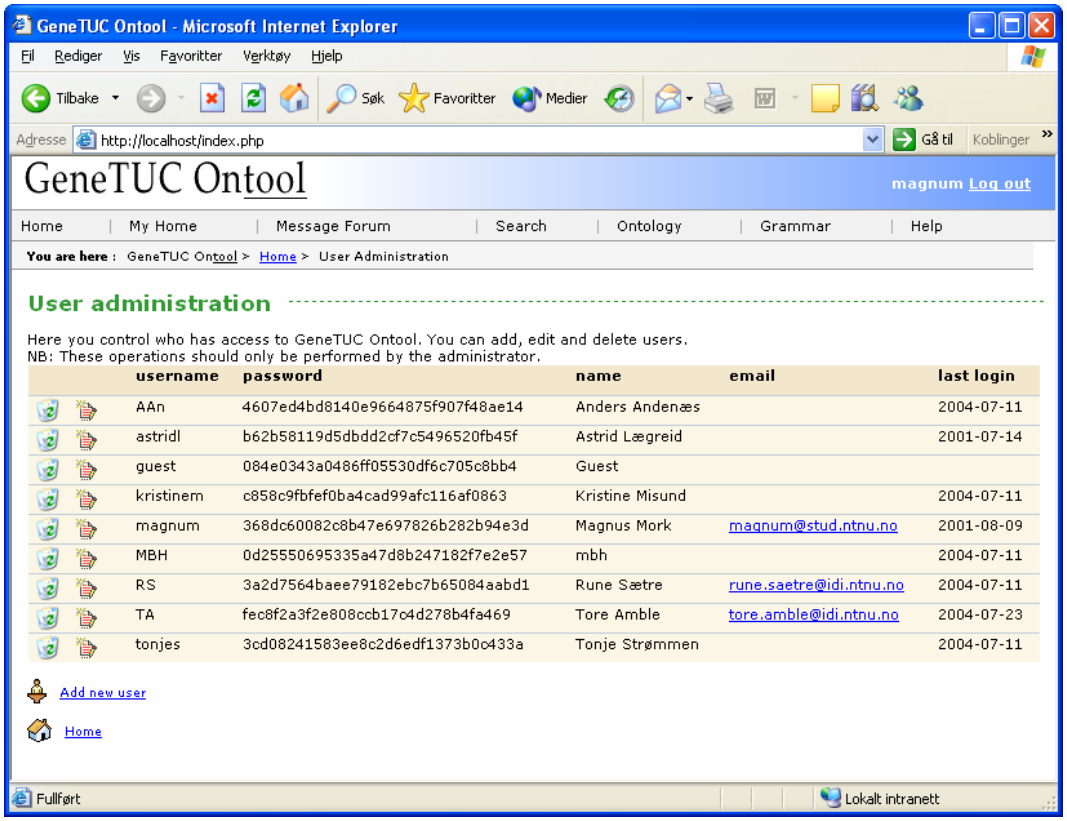


Figure 14.2 Ontool GUI - User administration

Figure 14.3 Usability evaluation of User administration

Assign terms to user

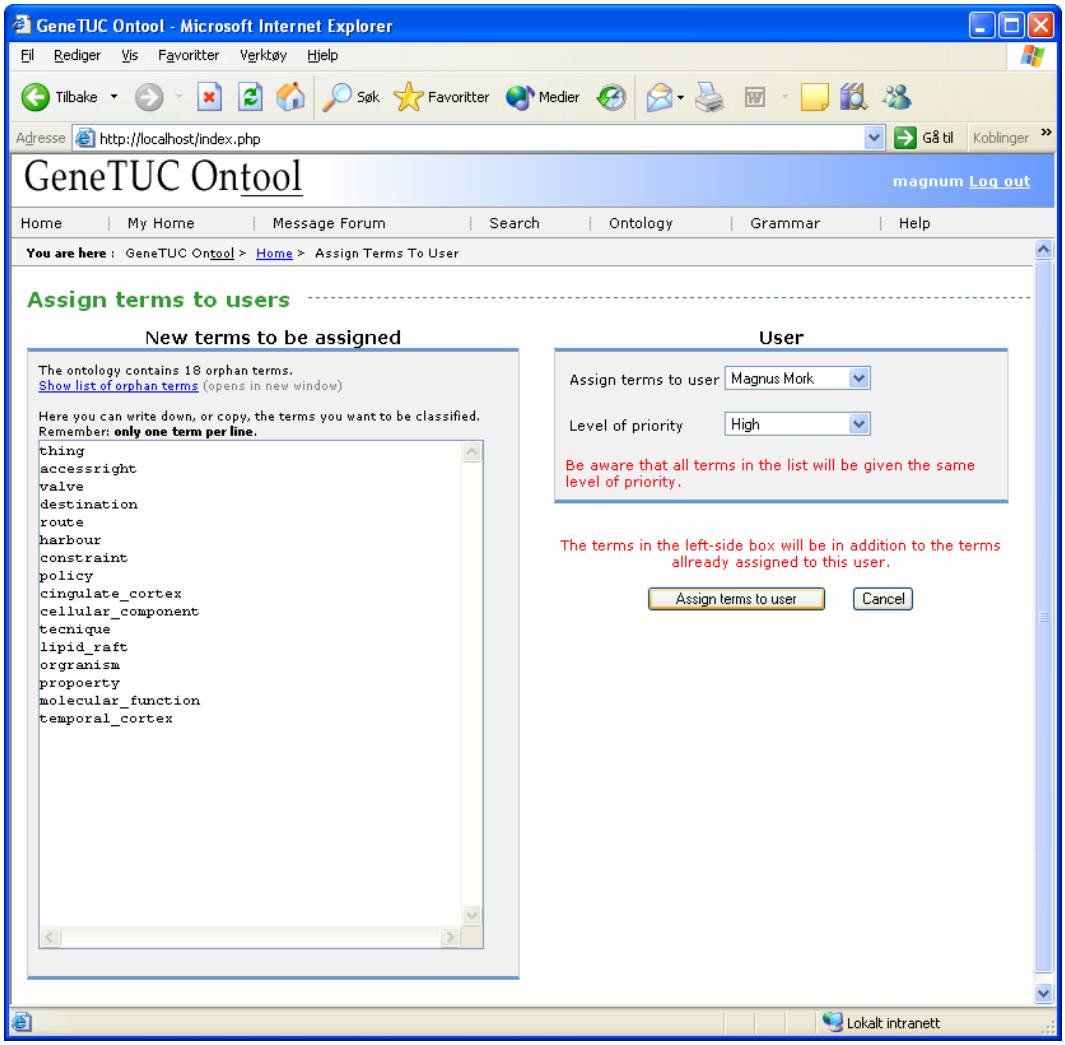


Figure 14.4 Ontool GUI - Assign terms to user

Figure 14.5 Usability evaluation of Assign terms to user

My Assignments / Add a relation

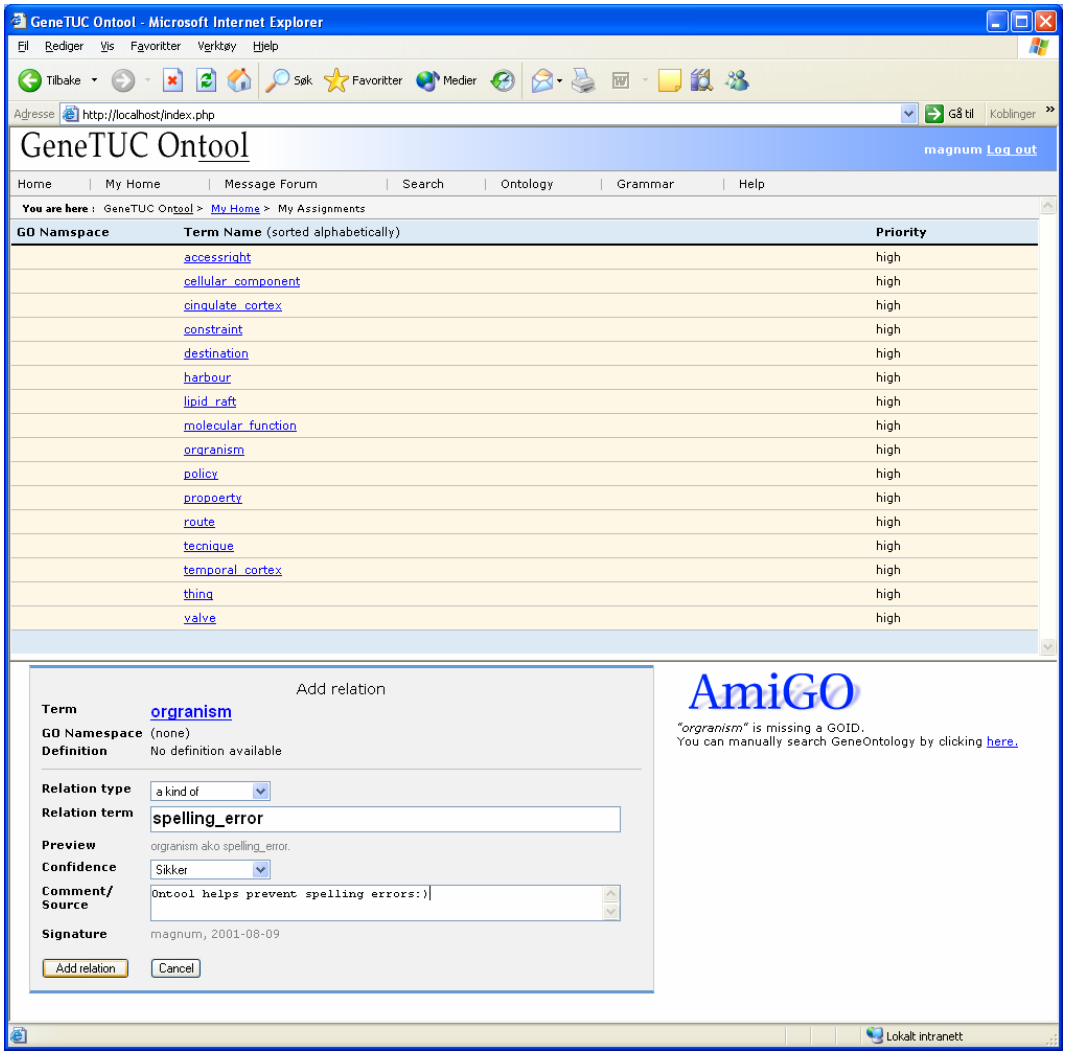


Figure 14.6 Ontool GUI - My Assignments / Add a relation

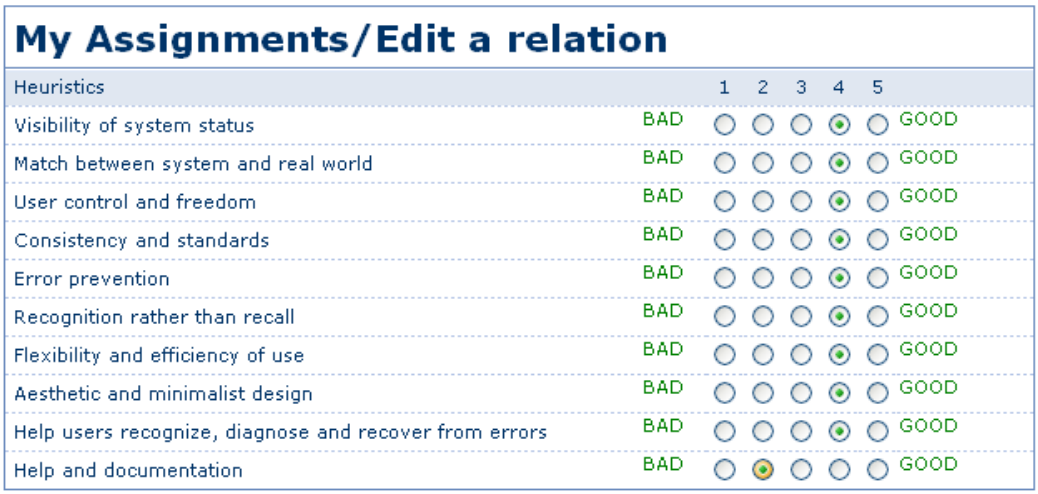


Figure 14.7 Usability evaluation My Assignments / Edit a relation

Show Term

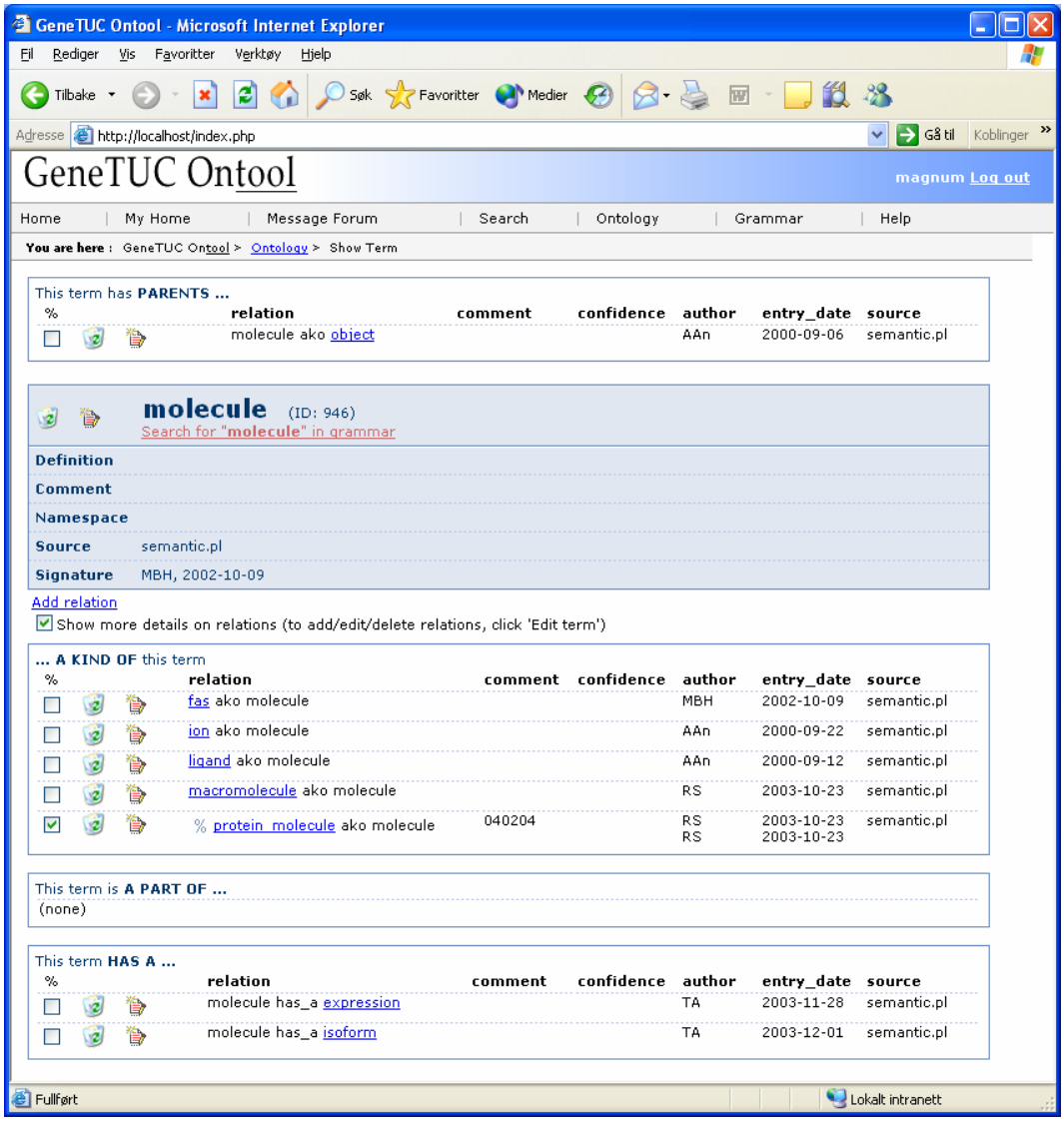


Figure 14.8 Ontool GUI - Show term

Figure 14.9 Usability evaluation Show term

Search Grammar

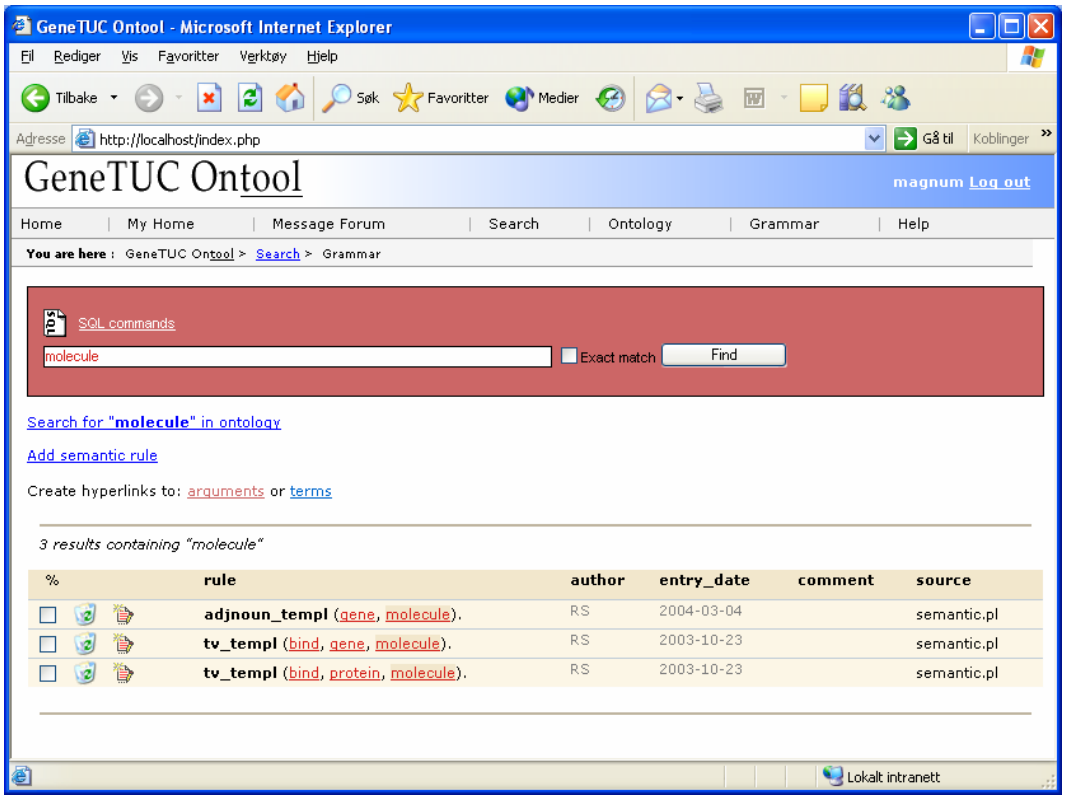


Figure 14.10 Ontool GUI - Search Grammar

Figure 14.11 Usability evaluation Search Grammar

Add / Edit a rule

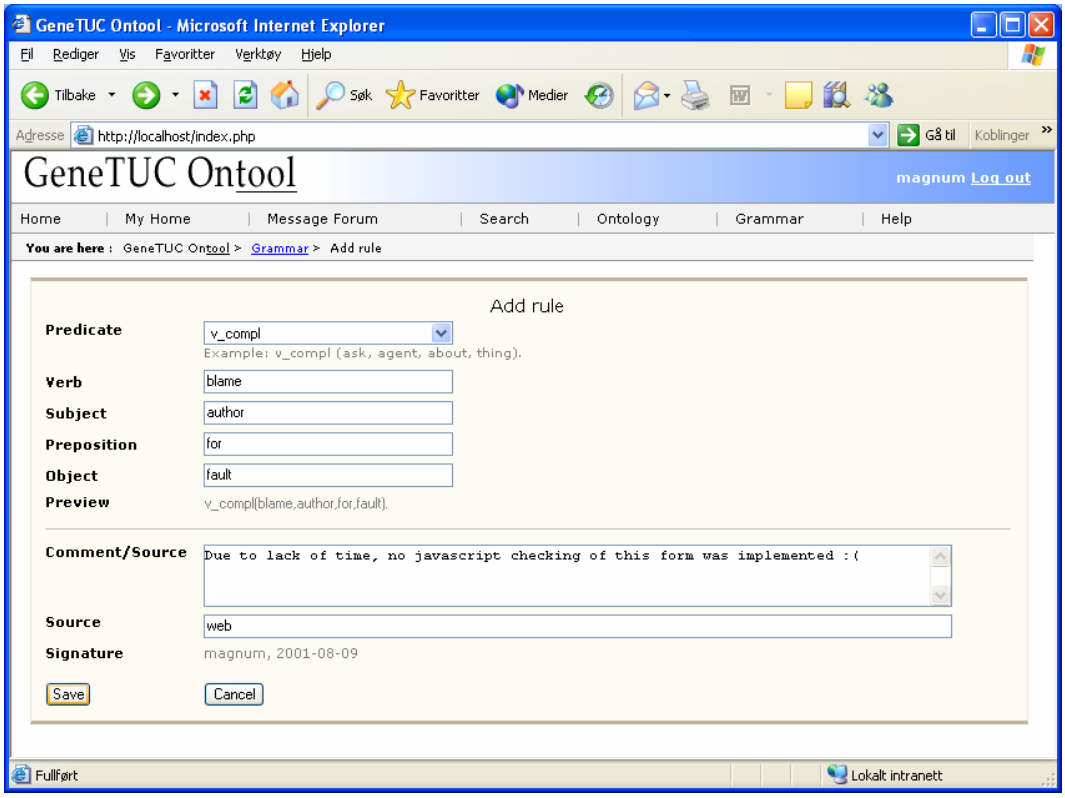
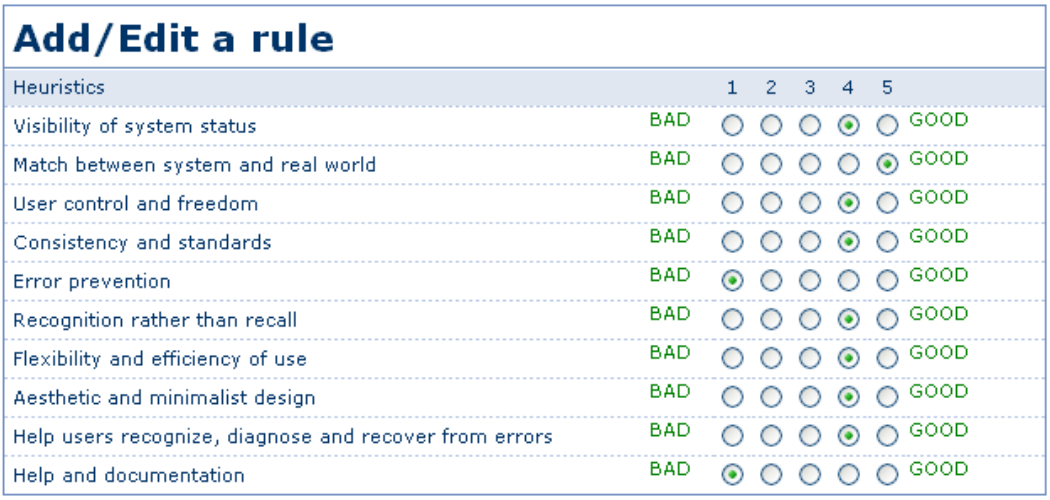


Figure 14.12 Ontool GUI - Add/Edit a rule



Add/Edit a rule

Heuristics		1	2	3	4	5	
Visibility of system status	BAD	○	○	○	◉	○	GOOD
Match between system and real world	BAD	○	○	○	○	◉	GOOD
User control and freedom	BAD	○	○	○	◉	○	GOOD
Consistency and standards	BAD	○	○	○	◉	○	GOOD
Error prevention	BAD	◉	○	○	○	○	GOOD
Recognition rather than recall	BAD	○	○	○	◉	○	GOOD
Flexibility and efficiency of use	BAD	○	○	○	◉	○	GOOD
Aesthetic and minimalist design	BAD	○	○	○	◉	○	GOOD
Help users recognize, diagnose and recover from errors	BAD	○	○	○	◉	○	GOOD
Help and documentation	BAD	◉	○	○	○	○	GOOD

Figure 14.13 Usability evaluation Add/Edit a rule

Part IV
Findings and Conclusion

15 Results and Contributions

15.1 The Ontool application

Part of the thesis consisted in specifying and implementing a web based database application for administrating the GeneTUC KB. The Ontool application was implemented according to the requirements given in chapter 11. Details on Ontool architecture and design is given in chapter 12.

To verify the functionality of the Ontool application, a series of system tests were carried out. The results of these tests will be discussed in chapter 16.3.

To measure the usability of the Ontool application, a series of usability tests were carried out. The results of these tests will be discussed in chapter 16.4.

15.2 Importing Gene Ontology

In accordance with the user requirements, the Ontool application has a built in function for importing information from the Gene Ontology knowledge base.

As a part of the thesis, the information relevant to the GeneTUC knowledge base was imported from Gene Ontology. The below table summarizes the results of the import process.

# terms imported from GO	17 142
# relations imported from GO	22 304

Table 15.1 Results from importing Gene Ontology

The importation drastically increased the size of the GeneTUC Ontology; extending it with an overall 1 263% (!).

As seen in chapter 6.2, the Gene Ontology knowledge base is fundamentally different from the GeneTUC KB. GeneTUC's KB is

organized as a heterarchical tree-structure where every term must be connected to the top-node with an ”a kind of” relation; either directly, or through a series of intermediate terms and relations. The Gene Ontology importation lead to an explosive increase in the number of orphan terms; going from a modest 18 orphans to a drastic 2108. In this respect, the aftermath of the importation is a pressing need for classifications. This was, however, anticipated by the GeneTUC team, and one of the reasons for developing the Ontool application.

15.3 Contributions

The main contributions of this thesis can by summarized as follows:

- Provides an overview of (1) the GeneTUC system and its underlying framework, (2) available technology, (3) Software development theory of Extreme Programming and (4) Usability- and usability testing.
- Provides a web based application, Ontool, which offer a (1) database backbone, (2) logic, and a (3) web user interface to the GeneTUC KB. Ontool can be ported to other systems built on the TUC framework.
- Provides detailed documentation of the Ontool development process.
- Provides results from usability tests on the Ontool application.
- Provides an evaluation on the Ontool application, and ideas for future extensions and improvements.
- Gene Ontology was successfully imported into the GeneTUC KB. This resulted in a 1 263% increase in KB size.
- The author has gained invaluable experience from leading a software development project, and communicating with the involved participants.

16 Discussion

This section provides a discussion of the choices made during this thesis, and the results that have been found.

16.1 Extreme Programming in this project

As discussed in chapter 8, the author chose to apply a formalized method of software development when working with this project. In the following the author will present his experience with the best practices of XP, along with an evaluation of his own work.

16.1.1 Execution of the XP best practices

Planning Game

The project was indeed driven by user stories, and plans were made based on the author's estimates of these stories. There were, however, no formal agreements on delivering date or functionality. There were no consequences for falling behind schedule; a new schedule was created in stead.

On-site Customer

The author had good contact with Rune Sætre during the project. Other *real live customers* were also kept in the loop, through monthly meetings.

Small Releases

Small releases were delivered to users for testing. When looking back, the author feels that this was not emphasized enough, and as a consequence it was not done in a high enough degree.

Simple Design

This practice will be discussed in detail in chapter 16.5.

Pair Programming

The author was alone when coding, and no control of code was carried out. When looking back, the author regrets not using the project supervisor, Rune Sætre, to overlook the source code. This could have been done once or twice during the project, to get feedback on code design, level of comments etc.

Test-driven Development

Extensive use of testing was used during implementation of the Ontool application. Both unit tests and acceptance tests were carried out, although the acceptance tests were not specified by the actual users, but by the author himself. When looking back, is becomes clear to the author that this could in fact have been done by the project supervisor.

Design Improvement/Refactoring

This practice was carried out with a certain level of success. The Ontool application uses a specific file (`genetuc_common_functions.php`) to hold functions used in multiple parts of the application. This counteracts duplication.

Continuous Integration

This was carried out with great success.

Collective Code Ownership

As the author was alone on the project, this practice does not apply.

Coding Standard

All code was written in a similar manner, to enhance readability and comprehensibility.

Metaphor

As the author was alone on the project, this practice does not apply.

Sustainable Pace

This practice was not carried out with success. Due to external circumstances and the author's wishes, the execution of the project was highly fluctuating. There were weeks with extensive use of overtime, and there were weeks with modest work.

16.1.2 Evaluation of the use of XP

It is the general opinion of the author that the use of XP is this project was only a mild success. This is not due to the XP practices alone, but rather the extent to which the author attempted to live up to the practices. The practices were always present in the head of the author, but as there was a lack of authority- and consequences, the author never fully committed to the methodology. The practices became guidelines, and nothing more.

The author feels that the introduction of an authority into the XP team and formalizing certain aspects of the development process at an earlier stage could have been of great help to the project.

16.2 Importing Gene Ontology

We have seen how the GeneTUC KB was extended by importing information from the Gene Ontology knowledge base. Whether this increase in KB size will result in better performance for the GeneTUC system is still unknown.

An initial test of the importation per se can be carried out, but the result will hardly carry any significance. The principal test of GeneTUC's performance should be carried out after the imported data has been completely adapted to the GeneTUC KB structure; in other words, when all orphan terms have been classified.

It is the hope and belief of the GeneTUC group that the importation and adaptation of the Gene Ontology knowledge base will lead to better performance of the GeneTUC system.

16.3 Ontool System test

The Ontool system tests were presented in chapter 13 along with the respective test results. Although all tests were given a "pass" verdict, the author would like to comment on the execution and evaluation of the tests.

16.3.1 Sub unit tests

Sub-unit tests (with F-#)	**Execution and evaluation**	**Result**
Logging on (F-1)	The test was passed with no special remarks.	Passed
User administration (F-2)	The test was passed with no special remarks.	Passed
Message Forum (F-3)	The test was passed with no special remarks.	Passed
Assign tasks to users (F-4)	The test was passed with no special remarks.	Passed
GeneTUC Settings (no F#)	The test was passed with no special remarks.	Passed
Show a personalized opening page, with "My Assignments" (F-5)	The test was passed with no special remarks.	Passed
Show Ontology as a hyperlinked tree structure. (F-6)	The test was given a "pass" verdict, although the implementation of the tree structure differ somewhat from the specified requirement. The tree structure only shows the "ako" relations, and so does not provide any information about terms which are connected by other relation types.	Passed
Show Grammar. (F-7)	The test was passed with no special remarks.	Passed

Search Ontology (F-8)	The test was passed with no special remarks.	Passed
Search Grammar (F-9)	The test was passed with no special remarks.	Passed
Edit terms (F-26,27,28)	The test was passed with no special remarks.	Passed
Edit relations (F-10,11,12,13)	The test was passed with no special remarks.	Passed
Edit rules (F-14,15,16,17)	The test was passed with no special remarks.	Passed
Show a chronological list of updates to the KB (F-18)	The test was passed with no special remarks.	Passed
Review classifications (F-19)	The test was passed with no special remarks.	Passed
View statistics about the KB (F-20)	The test was passed with no special remarks.	Passed
Import flat text files into KB. (also Gene Ontology) (F-21)	The test was given a "pass" verdict, although the author did not have time to verify that the import process was carried out without any form of errors. The import feature has been changed many times during implementation, and has been prone to errors. When carrying out the tests it seemed to be working.	Passed
The possibility to manually type in Prolog code that does not interact with the DB (F-22)	The test was passed with no special remarks.	Passed
Export KB to Prolog format. (F-23)	The test was passed with no special remarks.	Passed

Table 16.1 Comments on sub unit test results

16.3.2 System tests

System test # 1	The expert contributes to the contents of the KB
Description / Sub Unit tests	See sub test "Show Ontology as a hyperlinked tree structure.(F-6)"
Result	Passed

Table 16.2 Comments on system test#1 results

System test # 2	The administrator maintains the KB
Description / Sub Unit tests	See sub test "Import flat text files into KB. (also Gene Ontology) (F-21)"
Result	Passed

Table 16.3 Comments on system test#2 results

16.3.3 Non-Functional Requirement Tests

Quality requirement	**Description**
Performance	Start-up: The test was passed with no special remarks. Command execution: The test was passed with no special remarks.
Portability	The test was passed with no special remarks.
Robustness/ RELIABILITY	The test was passed even though the input was not specifically tested for correct type (number, text etc).

Table 16.4 Non-functional requirement tests for the Ontool application

The system passed all tests without any major flaws. The conclusion is that Ontool has passed the test process described in this chapter.

16.4 Ontool Usability

The Ontool application was developed for Microsoft Internet Explorer version 6, as this is the most commonly used web browser. However, the author performed tests on the Ontool application using a variety of client web browsers to ensure usability. Ontool was usable in all tested browsers, although the GUI varied slightly due to browser interpretation of HTML and CSS.

The Ontool GUI was designed to maximize usability of the application. The author considered several simpler GUI designs more in line with traditional research applications, where the functionality, not the usability and presentation, is in focus. It is the opinion of the author that the implemented GUI design is appealing to the user, and helps usability. Other people are allowed to have other opinions.

16.4.1 Comments on the usability tests

Usability test # 1 was carried out while Ontool was in an early prototype stage. The purpose was to get feedback on the chosen structure and design of the application.

The test results were positive, and so the author continued developing the application along the same lines as this early prototype.

Usability test # 2 was carried out when the Ontool application was stable and almost ready for delivery. The purpose was to get feedback on the usability of a few key functions of the application. The author had doubts about the usability of these functions, and so a formal usability test was the natural way to proceed.

The overall test results were positive, except for the "Help and documentation" heuristic. At the time of executing the usability test # 2, there existed no *Help* feature in the Ontool application. There existed a link to a help page, but no actual page. *All* participants found this link, clicked it, and got very disappointed when no help was provided.

When evaluation the usability on a task level the results were more or less as expected. Especially the *Error prevention* of the *Add/Edit a rule* task was expected, as no JavaScript `check_form()` function was implemented for this page.

The author would like to remind the reader that none of the participants in the usability test # 2 had any domain knowledge, and would not be the actual end users of the final Ontool application. All participants complained about the difficulty of setting the *Match between system and real world* and *Consistency and standards* heuristics. Also, all participants in both usability test # 1 and 2 were colleagues and friends of the author. This might bias the evaluation and the conclusions drawn from it.

16.5 Choice of design

The XP best practice on design, states: "*An XP team keeps the design exactly suited for the current functionality of the system. The requirements will change tomorrow, so the team only does what is needed to meet today's requirements*". During execution of the thesis project, the author has followed this principle.

The theory behind the practice sounds reasonable, and the author can believe that successful XP projects have been carried out using this principle. In the case of the thesis project, however, the author is critical to the implementation of the practice. It is the opinion of the author that this can be seen as a "lazy" strategy which tempts the XP team to renounce responsibility, and opt for a design with a short life span. This certainly has been the case in the thesis project. See chapter 8.2.1 for an argumentation of Extreme Programming in the thesis project.

During execution of the thesis project the author chose a procedural approach when implementing the application. This was done because the requirements were unknown, and the author lacked insight into ontologies and the GeneTUC KB.

As the project proceeded and the author gained the necessary insight, the cost of re-designing the application had risen, and in the end seemed too costly to consider for the author. The XP design practice lead the author into a "it works, so let us leave it like that" mentality. If given time to re-design Ontool, the author would choose an object oriented approach as this lies closer to the author's heart, and provides a cleaner, more lucid design. See chapter 17.1 for a re-design of the application.

16.6 Porting Ontool to another TUC system

Ontool is a domain independent application, and should be easy to port to other TUC systems. The author has not carried out such an operation, but provides some thought on how this could be done.

The key to porting Ontool is to set up a database identical to the `diplom` database which Ontool uses for the GeneTUC system. Then alter the `connect_db2()` function in the *genetuc_common_functions.php* file. The function can be altered so that it connects to the desired database based on the input from the login-form. In theory, this should be enough to start using the Ontool application with other TUC systems.

17 Future Work

As previously discussed, the implemented application includes only the basic features of a future complete system. This section details the future work that can be done to improve and extend the current implementation.

17.1 Re-designing Ontool

The author would strongly recommend a re-design of the Ontool application, as the current implementation lacks a neat and tidy design. The author would use an object oriented approach, make classes for `Term`, `Relation`, `Rule` and `Database`, as well as for `User` and `Message`. This approach would help maintain and extend the application, as it is designed in an orderly and tidy fashion.

Depending on available time, the author will start re-designing the application upon completion of the thesis report.

17.2 New features of Ontool

Searching

Future implementations might support *regular expression* searches, and an advanced search where some, or all, of the term-, rule- or relation can be searched.

User groups

Future implementations might differentiate on the set of possible tasks a user can perform, by not allowing an Expert to do Administrator tasks (add users etc). In other words; a user is categorized to a certain group and can only carry out actions according to this group.

Security

The current username/password check has limitations as browsers might reject cookies. Future implementations might implement other security measures.

User notification

Users might receive automatically generated emails with the most recent changes to the KB. Email notifications can also be given when a user has posted a message on the forum, which is relevant for other users.

Altering the database structure

Future implementations might provide means of altering the database structure through the web user interface. This will enable the user to alter the attributes to terms-, rules- and relations. When developing Ontool, the author used the *PhpMyAdmin* application for this purpose, and saw no need to implement these features.

File Upload

The current version of Ontool only supports importing of files which resides in a specific directory on the application server. Future implementations might support file uploads, so that users can upload text files from the web interface, and then import data from these files.

Export

The current version of Ontool only supports exporting to the Prolog file format, as this was the only format needed by the users. Future implementations might support exporting to other formats, as the user needs might change.

17.3 Ontool usability

When writing this, the author has had time to distance himself from the GUI of the Ontool application, and can look at the application with *new* eyes. The following table summarizes some of the author's *new* impressions, and ideas for future improvements to the usability of the Ontool application.

Show Term	It is the opinion of the author that this page is hard to use. The presentation is too unstructured, and the functionality provided is hidden.
Search GeneTUC	Changing the background color of the search frame, according to whether the user chose Grammar (red) or Ontool (blue) could help usability. An even better solution would be to make the search work in both the Grammar and Ontool, so that the user did not need to specify a search area.
Ontool menu	Remove the underlining on the "GeneTUC Ontool" in the menu, as this invites the user to click it as a hyperlink.
Help	Insert links to relevant help on all Ontool pages. The links could be in the shape of  images.

Table 17.1 Tips for improving Ontool usability

Bibliography

[1] George F. Luger. "Artificial Intelligence – Structures and Strategies for Complex Problem Solving", 4th edition. Addison-Wesley, 2002.

[2] Anders Andenæs. "GeneTUC. An NLP System for Biomedical Texts". Master Thesis, IDI, NTNU, 2000.

[3] Browser Statistics
URL http://www.w3schools.com/browsers/browsers_stats.asp.
Accessed: 2004-07-30

[4] W3C Homepage of CSS.
URL http://www.w3c.org/CSS.
Accessed: 2004-07-30

[5] Tore Bruland. "ExamTUC – A Simple Examination System in Natural Language". Master Thesis, IDI, NTNU, 2002.

[6] Gene Ontology Homepage.
URL http://www.gene-ontology.net.
Accessed: 2004-07-30.

[7] About Jakob Nielsen.
URL http://www.useit.com/jakob/.
Accessed: 2004-07-30.

[8] 10 usability heuristics of Jacob Nilsen.
URL http://www.useit.com/papers/heuristic/heuristic_list.html.
Accessed: 2004-07-30.

[9] Jakob Nielsen. "Guerilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier". Academic Press, Boston, 1994.

[10] Luke Welling. "PHP and MySQL Web development". Sams, 2001.

[11] Nils J Nilsson. "Artificial Intelligence: A New Synthesis". Morgan Kaufmann Publishers Inc, 1998.

[12] Natalya F Noy and Deborah L McGuiness. "A Guide to Creating Your First Ontology". Stanford University, Stanford, California.

[13] Open Biological Ontologies Homepage.
URL http://obo.sourceforge.net.
Accessed: 2004-07-30.

[14] Description on the Gene Ontologu OBO file format.
URL http://www.geneontology.org/GO.format.html.
Accessed: 2004-07-30.

[15] Ivan Bratko. "Prolog programming for artificial intelligence", 3rd edition. Addison-Wesley, 2001.

[16] J.Ross Quinlan. "A Formal Deductive Problem-Solving System". ACM Press, 1968

[17] Rune Sætre. "GeneTUC v2. A Biolinguistic Project, Next Generation". Master Thesis, IDI, NTNU, 2002.

[18] Homepage of Rune Sætre and the GeneTUC project URL http://www.idi.ntnu.no/~satre/genetuc/. Accessed 2004-07-30.

[19] Tore Amble. "The Understanding Computer – Natural language understanding in practice. Preliminary version". IDI, NTNU, 2002.

[20] Usability. URL http://atwww.hhi.de/USINACTS/tutorial/usabi2.html. Accessed: 2004-07-30.

[21] UsabilityTest URL http://www.userdesign.com/usability_uem.html. Accessed: 2004-07-30.

[22] Martin Fowler and Kendall Scott. "UML distilled: applying the standard object modeling language". Addison-Wesley, 1997.

[23] World Wide Web Consortium (W3C). URL http://www.w3c.com. Accessed: 2004-07-30.

[24] Cascading Style Sheets Homepage at W3C. URL http://www.w3c.com/css. Accessed: 2004-07-30.

[25] Document Object Model Homepage at W3C. URL http://www.w3c.com/dom. Accessed: 2004-07-30.

[26] WebHeuristics. URL http://user-experience.org/uefiles/writings/heuristics.html. Accessed: 2004-07-30.

[27] WebSurvey. URL http://news.netcraft.com/archives/web_server_survey.html. Accessed: 2004-07-30.

[28] Extreme Programming Homepage. URL http://www.extremeprogramming.org. Accessed: 2004-07-30.

[29] Paul Beynon-Davies. "DataBase Systems",3rd Edition. Palgrave Macmillan, 2004.

Appendix

A. Ontool installation guide

This document takes us through the installation process of the Ontool application. As described earlier, the Ontool system can be run on various platforms, with varying underlying software. It is not the goal of the author to write a manual for all possible environments, so general details on installing the underlying software is omitted. The focus of this document is to install the actual Ontool application, and to configure the environment to run with Ontool.

Ontool environment

The following components are necessary for successfully installing and running Ontool.

Computer	No definite hardware specifications are given as this is dependent on the chosen platform and software.
Internet access	The computer which will host the Ontool system must be connected to the Internet.
Web server with PHP	A web server must be running on the system. The web server must support PHP, and the PHP runtime engine must be installed and configured to run with the web server.
MySQL server	The computer must have a MySQL server running, or be able to access a MySQL server running on another computer.

Components necessary to run Ontool

Installing Ontool

Install Ontool by unzipping the Ontool ZIP archive to the desired directory. Be sure to maintain the directory structure from the zip file. If the application is not provided as a ZIP archive, but rather a set of files, just copy the whole file structure over to the destination directory.

Configure the web server so that it can access the directory where you installed Ontool (destination directory).

Writing permissions must be set for the following files, as Ontool uses these files for storing information on the file system.

Filename	Description
`genetuc_settings.txt`	Stores Ontool settings
`log/activity.txt`	Log of Ontool activity
`log/error.txt`	Log of Ontool errors
`semanticpl_freetext.txt`	Prolog code relevant for exporting the KB, and using it with the GeneTUC system.

Ontool files which need write-permissions

Configuring PHP

Make sure the `display_errors` flag in the PHP settings file is set to "Off", to avoid PHP error messages to interfere with the Ontool application layout.

" ; Print out errors (as a part of the output). For production web sites,
; you're strongly encouraged to turn this feature off, and use error logging
; instead (see below). Keeping display_errors enabled on a production web site
; may reveal security information to end users, such as file paths on your Web
; server, your database schema or other information.
display_errors = Off"

Setting up the database

Set up the Ontool database, by executing the SQL queries provided in the `/db/diplom.sql` file.

Connect the Ontool application to the database

The `connect_db2()` function in `genetuc_common_functions.php` must be altered. Change the `$mysql_user`, `$mysql_password` and `$mysql_host` to correspond to the MySQL configuration.

```php
/**
 * Connect to database
 *
 * This will connect the Ontool application to the database which runs on a MySQL server.
 * The $mysql_user, $mysql_password and $mysql_host must be set accoring to MySQL logon information
 *
 * @author Magnus Mork
 * @return resource Returns a MySQL link identifier on success, or FALSE on failure.
 */
function connect_db2(){
    $mysql_user="magnum";
    $mysql_password="";
    $mysql_host="localhost";
    $conn = mysql_connect($mysql_host, $mysql_user, $mysql_password)
       or die("Could not connect : " . mysql_error());
    mysql_select_db("diplom", $conn) or die("Could not select database");
    return $conn;
}
```

Edit this function to allow Ontool to connect to the database

If you have followed the above steps, and the underlying software is working correctly, you should be able to access Ontool by directing your web browser to the Ontool URL.

Log on to Ontool by typing in your username/password. For the purpose of evaluating the thesis, a user "guest" has been created, and a password is provided with the thesis documentation.

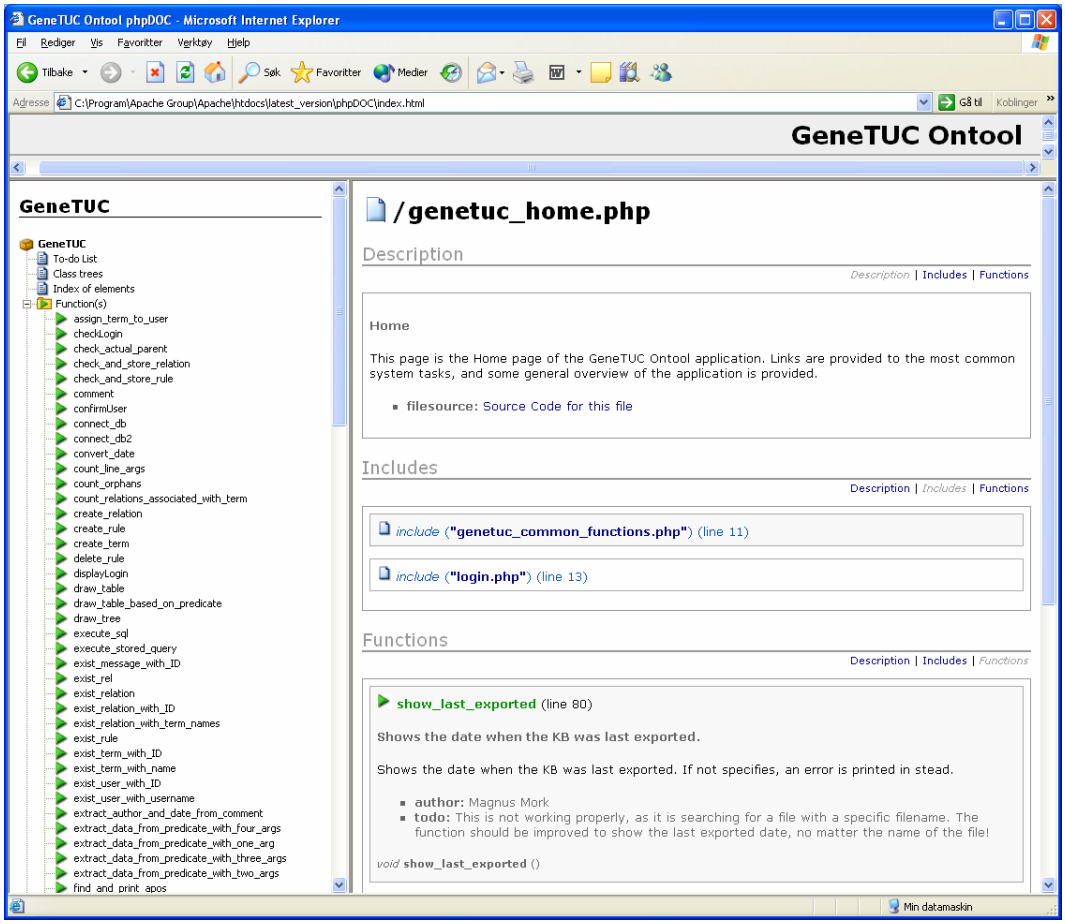
B. Ontool Help

The Ontool Help document is not provided with the thesis report as the application is due to change. Please refer to the *Help* feature of the Ontool application for the most recent help on the use of the application.

C. Ontool phpDOC

The author used the **phpDocumentor v1.3.0RC3** application to generate documentation of the PHP code. This documentation can be found in the `/Ontool_phpDOC` directory of the Ontool application.

The Ontool phpDOC is not provided with the thesis report, as this would probably more than double the number of pages to print. The author provides a screen grab of the documentation so that the reader can get an idea of the documentation structure.



GeneTUC Ontool PHP Documentation

D. Usability tips for Web pages

The author found this web-usability check list very useful. It is taken from a thesis on web usability and supporting software tools (http://www.tri.sbc.com/hfweb/brajnik/hfweb-brajnik.html).

consistency of presentation and controls

- *underline*: avoid mixing underlined text with underlined links
- *link label*: different links pointing to the same resource should have the same label
- *email label*: labels associated to a given email address should be consistent
- *color consistency*: colors used for background/foreground/links should be consistent among pages
- *background consistency*: background images should be consistently used
- *nav-bar consistency*: links included in navigation bars should be consistent among pages

adequate feedback

- *freshness*: pages should be time- and author- stamped

natural organization of the information

contextual navigation (in each state the required navigation options are available)

- *NOFRAMES validity*: NOFRAMES should be present and it should contain equivalent navigation options
- *link to home*: each page should contain a link to the home page
- *logical path*: each page should contain links to each intermediate page in the path connecting the page to the home
- *self-referential pages*: pages should not contain links to themselves
- *frame titles*: frames should set the "title" attribute
- *local links validity*: links that are local to the website should point to existing resources
- *external links validity*: links to external resources should be periodically checked

efficient navigation

- *site depth*: the number of links that need to be followed from home page to other pages should not exceed a threshold
- *table coding*: table components should have explicit width and height
- *image coding*: images should also have explicit width and height
- *download time*: pages should download within given time threshold

- *recycled graphics*: images used in the website should be shared (so that browsers can cache them)
- *hidden elements*: pages should not contain elements that cannot be shown (like maps not associated to any image)

clear and meaningful labels

- *informative link labels*: links pointing to heavy/plug-in dependent resources should specify that in the label
- *explicit mailto addresses*: labels of "mailto:" links should contain the actual email address
- *missing page title*: pages should have a title
- *table headers*: tables should have headers and summaries
- *form prompts*: within forms, text input fields should have a label

robustness (of the site with respect to the technology used by users)

- *browser compatibility*: HTML code should not use proprietary structures
- *safe colors*: page elements should use web-safe colors
- *link targets*: avoid "_blank" target in frames; use correct targets for links leaving the frames
- *HTML validity*: only standard HTML code should be used
- *portable font-faces*: standard font faces should be used in addition to desired ones
- *color contrast*: background and foreground colors combinations should provide sufficient contrast

flexibility

- *image ALT*: images should have alternative textual descriptions
- *other media ALT*: videos, audios, applets and other objects should have alternative textual descriptions
- *imagemap links*: links embedded in images should be available also in textual format
- *auto-refresh*: duplicate auto-refresh links in the page body (both forward and backward ones)
- *forced downloading*: links embedding an image in their label cannot be followed without downloading the image
- *tables/frames/font resizing*: relative sizes should be used

support of users' goals

- *form coding*: forms should have "submit", "reset" buttons

maintainability

- *relative links*: URLs that are local to the website should be relative

other

- *spelling*: spell-check the content of pages
- *different media*: report on the number of different media that are used in pages/website

- *keywords/description*: pages should have appropriate META information to be searchable by search engines
- *site popularity*: how many other websites point to the one under analysis
- *marquee,blink*: avoid animated features

E. Gene Ontology

The ontologies

The three organizing principles of GO are *molecular function*, *biological process* and *cellular component*. A gene product has one or more molecular functions and is used in one or more biological processes; it might be associated with one or more cellular components. For example, the gene product cytochrome c can be described by the molecular function term electron transporter activity, the biological process terms oxidative phosphorylation and induction of cell death, and the cellular component terms mitochondrial matrix and mitochondrial inner membrane. Before we go any further, here are some definitions that should help you to distinguish a gene product from what it does.

Molecular function

Molecular function describes activities, such as catalytic or binding activities, at the molecular level. GO molecular function terms represent activities rather than the entities (molecules or complexes) that perform the actions, and do not specify where or when, or in what context, the action takes place. Molecular functions generally correspond to activities that can be performed by individual gene products, but some activities are performed by assembled complexes of gene products. Examples of broad functional terms are catalytic activity, transporter activity, or binding; examples of narrower functional terms are adenylate cyclase activity or Toll receptor binding.

It is easy to confuse a gene product with its molecular function, and for that reason many GO molecular functions are appended with the word "activity". The documentation on gene products (above) explains this confusion in more depth.

Biological process

A biological process is accomplished by one or more ordered assemblies of molecular functions. Examples of broad biological process terms are cell growth and maintenance or signal transduction. Examples of more specific terms are pyrimidine metabolism or alpha-glucoside transport. It can be difficult to distinguish between a biological process and a

molecular function, but the general rule is that a process must have more than one distinct steps.

A biological process is not equivalent to a pathway. We are specifically not capturing or trying to represent any of the dynamics or dependencies that would be required to describe a pathway.

Cellular component

A cellular component is just that, a component of a cell but with the proviso that it is part of some larger object, which may be an anatomical structure (e.g. rough endoplasmic reticulum or nucleus) or a gene product group (e.g. ribosome, proteasome or a protein dimer).

The above text was cut from the Gene Ontology web page; http://www.geneontology.org/GO.doc.html.