Support for Mobile Software Processes in CAGIS

Alf Inge Wang

Dept. of Computer and Information Science,
Norwegian University of Science and Technology (NTNU),
N-7491 Trondheim, Norway,
Phone: +47 73594485, Fax: +47 73594466,
Email: alfw@idi.ntnu.no
Web: http://www.idi.ntnu.no/~alfw

Abstract. This paper describes a prototype for supporting distributed, mobile software processes. The prototype allows instantiated process models to be distributed in different workspaces, and have mechanisms to allow parts of the process to be moved from one workspace to another. The paper outlines the main concepts, a process modelling language and tools to support distributed, mobile processes. Further, we discuss problems and possible solutions for our prototype, and some experiments are also outlined. This work has been carried out as a part of a project called CAGIS, described in the introduction of the paper.
Keywords: Mobile software process, Process Centred Environment, Workflow tool, Process Modelling Language, Web, XML, CGI, Software agents

1 Introduction

For many years, most process centred environments (PCEs) have made the assumption that one centralised process model is needed to represent the whole software process. Since the introduction of the Internet, more and more organisations work in a distributed way in heterogeneous environments. Distributed organisations must cope with management of people working in different places, on different times, with different tools and on different processes. For most traditional PCEs, it is impossible or at least very hard to model processes with a highly distributed and heterogeneous nature. In addition, when the organisation is divided into smaller, autonomous sub-organisations, it is impossible to use one centralised model to reflect software processes with the scope of the whole organisation. It is unthinkable, that one model, often managed by top-level management, shall represent all autonomous groups. The process model must reflect the organisation and thus be distributed into smaller autonomous parts [9, 12].

In 1986-1996, our research group, managed by Reidar Conradi, worked on a PCE prototype called EPOS [5]. EPOS was an advanced environment for managing software processes as well as software artifacts through various tools. In 1997, a project called *Cooperative Agents in Global Information Space* (CAGIS) [11] was started. One of the main goals of the CAGIS project was to see how heterogeneous, distributed, cooperative work could be supported. The first part of the project identified requirements for

how to support software process in a global information space. We soon found that our traditional EPOS environment could not fulfil requirements like openended-ness, distributed processes, heterogeneous tools and dynamic changing and movement of fragments of the process (*process fragments*). All these characteristics can be found in what we call Cooperative Software Engineering (CSE) [16]. EPOS suffered from being too centralised, too static and too closed a system to support CSE. The process models could only be changed in each workspace, and coordination between workspaces was not sufficiently supported. Thus a strict top-down approach changing the process model was enforced. In reality only upper management could evolve the process model.

To put this paper in the right context, we give a short review of an overall architecture, to which this work contributes. In [16], we presented our CAGIS multi-agent architecture for cooperative software engineering (see figure 1). The architecture consists of four components:

1. **Agents**: Agents are set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to environment, as well as pro-activeness. There are three main types of agents: (1) *Work agents* to assist in local software production activities, (2), *Interaction agents* to assist with cooperative work (such as communication, coordination, mediation and negotiation) between workspaces, and (3) *System agents* to give system support to other agents. Interaction agents are mobile agents, while system agents and work agents are stationary.
2. **Workspaces**: A workspace is a temporary container for relevant data in a suitable format, together with the processing tools. It can be private as well as shared.
3. **AgentMeetingPlace (AMP)**: AMPs are where agents meet and interact. AMPs provide agents with support for doing efficient inter-agent communication. An AMP is therefore a “special” workspace for software agents.
4. **Repositories**: Repositories can be global, local or distributed, and are persistent. Workspaces may check in and out information from repositories.

The architecture is implemented in Java, KQML is used for agent communication and IBM Aglets [8] are used to support mobile agents [10]. In figure 1, arrows between agents indicate inter-agent interaction. The arrows related to the *monitor agent*, describe that this agent is logging events in the two workspaces and the AMP, and store event information in the global repository. The *mediation agent* uses this information, retrieved from the global repository, to support the inter-agent negotiation process.

In the CAGIS CSE architecture, interaction agents perform all collaboration between workspaces. An AMP is the neutral meeting point for agents from different workspaces, and it supports the inter-workspace process. Within a workspace, a simple PCE or a workflow tool can be used to give support for the local process. Since the local process does not involve much coordination between involved actors (interaction agents are taking care of this bit), the local process becomes relatively simple. If a project manager wants to assign a specific job to a project member, interaction agents are used to find available human resources. When the available project member is found, the project manager can use agents to give this person a description of what to do, as well as a

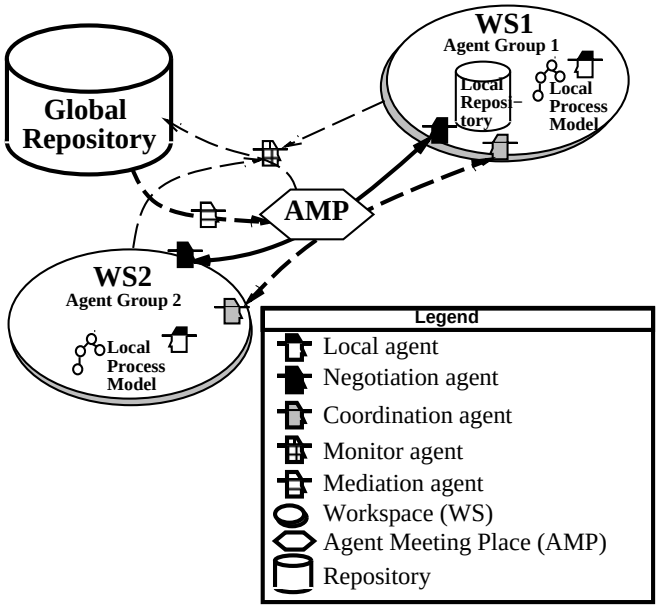

Fig. 1. MAS-based architecture for cooperative software engineering

local process model (telling him how to do it). This process model will then be moved from the project manager's workspace to the project members workspace.

This paper presents a prototype of a simple PCE/workflow system allowing a process model to be distributed on several workspaces and where parts of the process (process fragments) can be moved between workspaces. Remember that this prototype is not intended used as a stand-alone system, but it is a part of the CAGIS multi-agent architecture for CSE. Our prototype only supports simple and straightforward processes, since the interactive agents in the enclosing CAGIS architecture will take care of the more advanced processes.

The rest of the paper is organised as it follows. Section 2 present related work on distributed PCEs, and mobile workflow. Section 3 presents the main concepts of our approach. Section 4 briefly outlines some preliminary experiences, and discusses problems and possible solution for our prototype. Section 5 concludes the paper.

2 Related work

Within the Software Process community, the research area on how to support distributed and heterogeneous software processes has recently been popular topic. In [13], Tiako outlines how to model federation of Process Sensitive Engineering Environments (PSEEs). In such federation, large development projects can be realized by dividing the

whole process in to smaller pieces assigned to distributed teams. The teams work on their own processes using their own PSEEs. Tiako describes a federation process support architecture that aims to support not only the enactment of processes, but also their definition by composition, decomposition and/or federation.

In [2], Ben-Shaul et al. propose the Oz approach to provide a federated PSEE by composing different instances of local PSEEs. Each instance is devoted to support the development process executed by a single organisation. The different PSEEs run autonomously according to their own processes. Interaction of several PSEEs is accomplished through a common activity called *summit*, where one site acts as a coordinator and receives all needed data from other sites. The result is sent back from the coordinator to all involved sites.

In [1], Basile et al. take Oz as a starting point to provide federated PSEEs, and allows several inter-organisation policies to be implemented and combined. A set of basic operations is used to specify any inter-organisational policy (e.g., one operation for searching the physical location of a site, one operation for requesting execution of one service on another site etc.).

In [4], Bhattacharyya and Osterweil address the problem of giving decision support on moving and relocate process fragments to users. When a user wants to go mobile, a request is sent to a relocation request analysis engine (RELOCATE). RELOCATE receives three types of data: Current process, network configuration and user request. The network configuration data is used to compute what the performance will be if the user goes mobile. The process information expresses the process structure (static information) and the process execution state (dynamic information). RELOCATE will produce an answer to the user as well as modified process information. The RELOCATE engine can be given an entire software process and asked to come up with an ”efficient” allocation of process fragments to users - in effect, producing a new, modified version of the software process. It can also be asked to deal with problems regarding specific aspects of mobile computing (e.g., use a laptop computer with a low processing speed).

In [18], Yoo and Lee describe a mobile agent platform for workflow systems called X-MAS (proXy acting Mobile Agent Systems). Here, the workflow system using the X-MAS mobile agent platform has a centralised coordinator. The workflow model (process model) is defined centrally in a workflow definition tool. The workflow management engine realizes workflow instances as mobile agents by asking the mobile agent platform to create them. If there are any time-constraints of agents, this information is stored in an agent manager in the agent execution engine. The mobile agents (workflow instances) may move from host to host, and interact with other entities as users, databases, and applications. A worklist handler in each location server enables mobile agents to run applications and interact with humans. When an agent is finished with his job and has come back, the workflow management engine stops the workflow instance of that agent. X-MAS is implemented in Java and Remote Method Invocation (RMI) in Java is used to implement agent mobility.

In [7], Jing et al. address how to adopt workflow systems to support mobile environments. For many companies, the attraction of mobile computing comes from possibly

large productivity gains in the out-of-office workplace. There is a need to give process support for this kind of mobile and dynamic environments, and mobility must be supported in workflow management systems. Such systems must deal with mobile resources (equipment as well as human), support for location-independent activities (work at home or anywhere), as well as location-dependent activities (need to go to a specific place, for instance to deal with a customer etc.). Mobile resource management needs to efficiently track resources, status of mobile resources, and the assignment of resources to work activities. This means dealing with problems regarding workflow resources that are not always connected and that they can change status not being connected. Since resources move around, synchronisation of workflow information can also cause some problems.

The first three papers presented in this section (Tiako [13], Ben-shaul [2], and Basile et al. [1]) discuss how to support federation of PSEE. Our paper does not discuss federation in particular, but touches issues such as local autonomy and distribution of process fragments. None of these papers look into how to support mobile software processes. Bhattacharyya and Osterweil [4] however, describe how to analyse the impact of relocation of a process fragment. We only describe the mechanisms to provide mobile process fragments, and does not provide advanced tools for analysing the impact. The two last papers described in this section, discusses mobile workflow from two different perspectives. Yoo and Lee [18] provide mobile workflow by making the process instances mobile agents, while Jing et al. [7] addresses issues for how to adopt workflow systems to support mobile environments. The former is opposite to our approach because we enables mobility of process fragments, and uses stationary process servers to execute distributed process fragments. The latter is more general and touches issues that are discussed in our paper such as mobile resource handling.

3 CAGIS Mobile Software Process Approach

This section describes the main concepts, the process modelling language (PML), how to move process fragments, and the tools to support mobile software processes.

3.1 Main Concepts

It is not so hard to see the similarities between process modelling and software programming. There are also some PMLs that are very similar to general programming languages (for instance the object-oriented text-based PML in ProcessWeb [17]). A running process model (instantiated model) can in this analogy be viewed as a running program. Traditional programs can only execute on the same machine and cannot be changed at runtime[1]. However, if the program is a mobile software agent, it is possible for the program to move between different machines and to change its properties during

[1] There are programming environments that allow programs to move and change during execution such as SmallTalk- and Java-programs

its lifetime. Our idea is for process models to have mobility as mobile software agents. This means that it is possible for process fragments to move between workspaces during enactment, as well as to evolve process fragment instances.

In our prototype environment, a process model can be distributed as shown in figure 2. The process model consists of several activities that are linked together. The process model is not represented as one centralised model, but is distributed to different workspaces. Within the workspaces, there is local autonomy for the local process models. This means that people working in a workspace can change their own process model. It is the links between activities that tie the process model together (also activities in different workspaces). A link between two activities describes which activity to be executed first (pre-order), and the prelink and postlink tags are used to describe a link in our PML (see section 3.2). To go from one activity to another (related with a link), a user explicitly tells the workflow tool that the first activity is finished by pushing a button in the workflow tool.

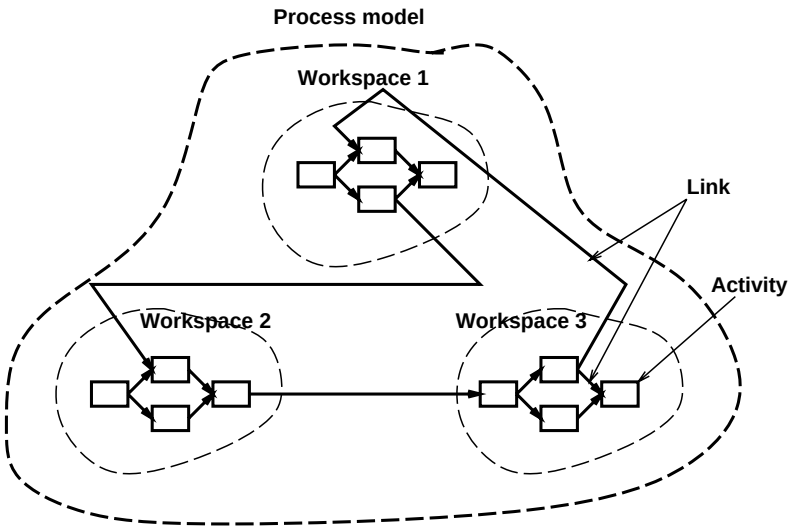

Fig. 2. Composition of a process

The smallest building block in our PML is an activity. All activities are represented as individual model objects, which are linked together in the same way as web pages on the Internet. Another central concept of our prototype environment and PML is *process fragment*. A process fragment is one or more activities that are linked together in a workspace. For example, the four activities in workspace 1 in figure 2, can be a process fragment. But a process fragment can also be any combination of related activities as shown in figure 3. The term process fragment can therefore be used to name the group of activities that are moved between workspaces. Since we use eXtended Markup Language (XML) [14] to wrap the process information being moved between workspaces,

we have defined the tag $< processfragment >$ to define the context for linked activities. More about this in section 3.2.

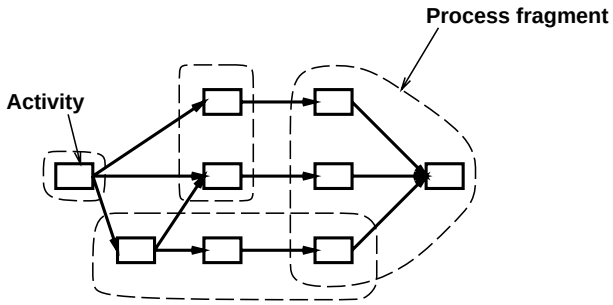

Fig. 3. The concept of process fragment

3.2 The Process Model Language

As mentioned above, an activity is an atomic building block in our PML. A process (fragment) is modelled as a collection of related activities. To specify an activity in our PML is similar to creating a small web page. Information is structured in XML, using $< tags >$ to specify valid syntax. The application interpreting the PML defines the semantic of the language. In figure 4, the Data Type Definition (DTD) for the PML is described. The DTD describes the elements needed to describe a process fragment and how these elements are structured. Note that '+' is used to describe one or more elements, '*' is used to describe none or more elements, while '?' states that there can be none or one element. *PCDATA* is used to specify the type of data tags, and means that these tags are of type parsed character data (text).

In our PML (see figure 4) a process consists of one or more process fragments, and a process fragment consists of one or more activities. A process is identified by a $< name >$, and a process fragment is identified by a $< name >$ and a $< workspace >$. Every activity in a process model defines a unique identifier by combining the $< workspace >$ defined for the process fragment and the $< name >$ of the activity. In order to have a unique identifier in a distributed environment, a URL is used as the workspace path. The $< prelink >$ tag defines the preconditions for an activity. This activity cannot start before the activities listed in prelink (listed as unique identifiers) are finished. The $< postlink >$ will define what activity(ies) to execute next. The *code* tag is used to specify the HTML code related to the activity. This code may range from a simple descriptive text, to the definition of a complex interaction via a Java applet. An activity can have three different states: *Waiting*, *Ready*, and *Finish*. The state *Waiting* indicates that the activity is waiting on one or more activities to finish before it can be executed (specified in $< prelink >$). The state *Ready* indicates that the activity is ready to start.

```
<?XML encoding=''UTF-9''?>
<!ELEMENT process (name,
                  (processfragment)+>
<!ELEMENT processfragment (name,
                           (workspace),
                           (activity)+)>
<!ELEMENT activity (name,
                   (workspace),
                   (prelink)*,
                   (postlink)*,
                   (state)?,
                   (due)?,
                   (feedback)?,
                   (description),
                   (code)*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT workspace (#PCDATA)>
<!ELEMENT prelink (#PCDATA)>
<!ELEMENT postlink (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT due (#PCDATA)>
<!ELEMENT feedback (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT code (#PCDATA)>
```

Fig. 4. XML Document Type Declaration of our PML

When a user is finished working with an activity, (s)he explicitly notifies the workflow tool that the activity is *Finished* by clicking on a button in the agenda tool (see section 3.4). The *feedback* tag is used to specify whether an activity is a part of a feedback loop or not. If an activity is modelled as a feedback activity, it can only have two prelinks, where one of the prelinks represents the feedback loop. A feedback activity is activated **if one** of the prelinks has the state *Finished*. This is different from an activity without feedback, where **all** prelinks must have the state *Finished* before it can be activated.

Figure 5 illustrates an example where the activity *compile* in workspace *Kramer* must wait for activities *code* and *read document* in workspace *Elaine* to finish, and the activity *build* in workspace *George* will be the last activity to be executed.

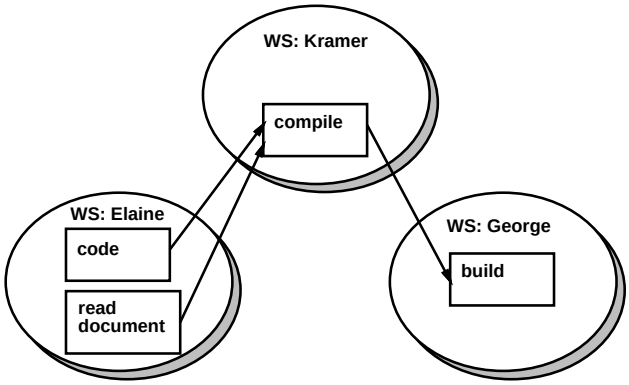

Fig. 5. Example with linked activities in several workspaces

Figure 6 shows how the dependencies between the activities are specified in our PML (XML syntax) for the example shown in figure 5. Since the activity *compile* (in the example above) has more than one $< prelink >$, it cannot be executed before *all* activities specified as $< prelink >$ are finished.

```
<Activity>
<Name>compile</Name>
<Workspace>Kramer</Workspace>
<Prelink>Elaine/code</Prelink>
<Prelink>Elaine/read document</Prelink>
<Postlink>George/build</Postlink>
...
</Activity>
```

Fig. 6. An example of use of $< prelink >$ and $< postlink >$ tags

When modelling a process in our PML, it is possible to either write the process model as an XML-document, or it is possible to use a form-based web-tool to create the model. When instantiating the process model, all activities are registered in a process server (see section 3.4) with an initial state, and XML-files defining each activity will be created in the specified workspace. It is not necessary to define the whole process model at once, but you can incrementally add process fragments when desired. The definition of an activity can be modified directly by simply editing the XML-file for this activity in its workspace (you can not change the state). For instance to change the activity *code* (in most cases simply HTML describing what to do), will not affect other activities and can be changed directly. However if *prelinks* and *postlinks* are changed, this will affect other activities. Such changes should not be executed, before the effect of the change can be analysed. A simple analysis would be to check that the activities described in the prelinks and postlinks tags are activities registered in the process server. A more advanced analysis would be to go through the whole activity-network to detect live-locks, dead-locks etc.

3.3 Moving Process Fragments

When a process fragment is moved between workspaces at the same process server, the $< workspace >$ tags are changed while the rest of the information are left unchanged. Then the associated XML-files are moved between the workspaces. The state of the activities in the involved process fragment will be unchanged. If a process fragment is moved between two process servers (two sites), this is done as illustrated in figure 7. Let's say that a process fragment in workspace 1 (WS1) at Site 1 is going to be moved to workspace 4 (WS4) at Site 2.

- **Step 1:** Process server 1 will call a CGI-script at Process server 2 with some parameters (e.g., *http://www.processserver2.no/movefragment.cgi?parameters*). The parameters will contain a reference to the workspace in Site 2 the process fragment is going to be moved to (e.g., *workspace=WS4*), a URL to the XML-file representing the process fragment (e.g., *xmlfile=http://www.processserver1.no/WS1/processfragment1.xml*), and possibly a URL to other files related to the process.
- **Step 2:** Process server 2 will send a request for downloading the XML-file and other related files from Process server 1 using the HTTP-protocol.
- **Step 3:** Process server 2 receives the XML file and other related files, and instantiate the process fragment in the process server and WS4, if there are no conflicts.
- **Step 4:** Process server 1 removes the process fragment from WS1 and removes all registered activities in the process fragment from the process server. Note that the state of the activities in the process fragment is also moved from Process server 1 to Process server 2.

There is also another way of moving process fragments and files between workspaces (also between sites). Our own CAGIS multi-agent architecture for cooperative software

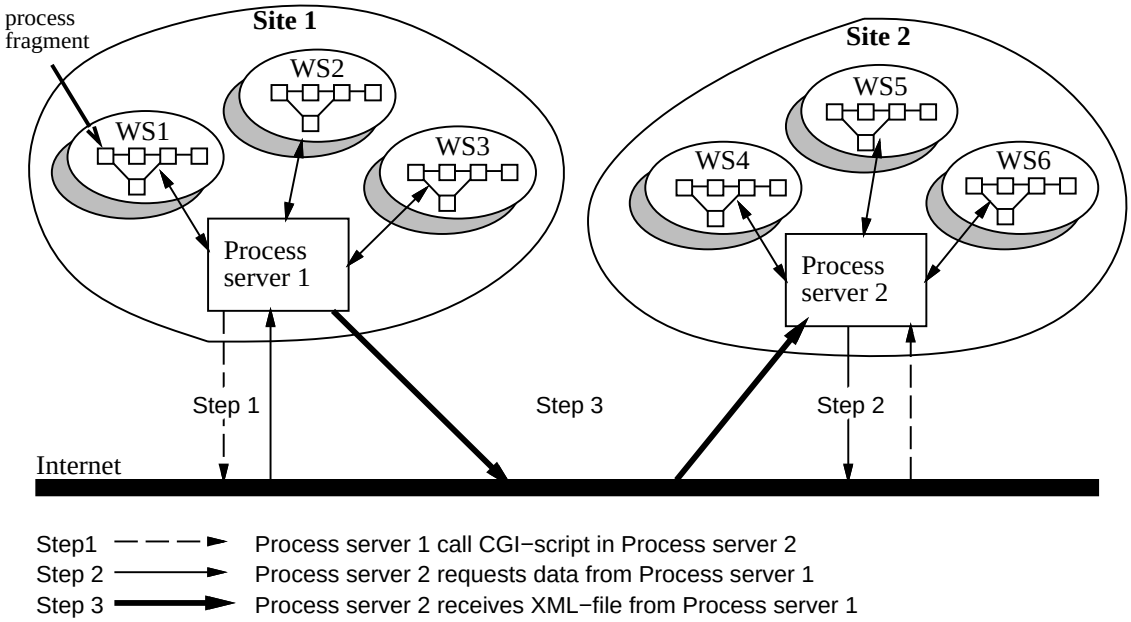

Fig. 7. Moving process fragments between different process servers

engineering described briefly in section 1 can be used for this purpose (see also [16, 10] for details). Coordination agents are used to transport process fragments represented in XML and other related files between workspaces. Other agents can also be used to support the more unpredictable and dynamic aspects of the process, e.g. negotiation about resources. Negotiation agents can also be used to solve arguments when people in a workspace reject process fragments moved from another workspace.

Since our process server has an open interface through CGI, other infrastructures can also be used for moving XML-files between workspaces. BSCW [3] is one example of such a system.

3.4 The Process Support Tools

Our system is programmed entirely in Perl and a web-interface is provided through a CGI. The system consists of four main components:

– **Process server/engine** The process server has three main tasks. *First*, it manages the state information of activities. Through CGI, the state of an activity can be changed. Synchronisation of an activity having several prelinks, is done by checking if all prelinks have the state *Finished* before the activity is activated. The process server also manages information about activity iterations. *Second*, the process

server manages registration of new activities as well as unregistration of existing activities (also during enactment). Registration and unregistration of activities are activated through the CGI-interface. *Third*, the process server manages moving process fragments to other process servers. Through CGI, other servers can ask to move process fragments to them. The registration and unregistration of activities described above is used to facilitate movement of process fragments. The process server does not make any decisions for when process fragments are going to be moved or not. These decisions are managed by a GlueServer and specified in a GlueModel [15]. The GlueModel defines rules for how workspaces and sites shall collaborate.

- **Process modeller** A web-client for incrementally writing process models This is a simple web-interface where it is possible to enter the process model, activity by activity, to view, modify and remove existing activities. There is also support for making it easier to model activities that are linked sequentially, by automatically filling in information into the form.
- **Agenda manager** A web-client that provides an agenda to users or groups of users. Through the agenda manager, you can choose what activities to execute next and navigate through the process. Users are separated through the workspace mechanism. A workspace can be for a single user or for a group. The agenda manager also is used to visualise activities.
- **Monitor** This is an administrative tool that makes it possible to monitor the state and the progress of a process. It is also possible to change state information of the process at runtime using this tool.

In figure 8, a screenshot of the three different process tools are shown. In the upper left corner, the **Process modeller** tool is shown, and to the right we can see the window of the **Monitor** tool. The two windows below are both from the **Agenda manager** tool. To the left, the agenda for the workspace *review/author* is shown. To the right, the activity *Edit paper* is shown.

4 Discussion and evaluation

The prototype is yet simple and does not provide much advanced functionality. However we have gained some experiences based on modelling different processes and letting user unfamiliar with process modelling try out the prototype. The first we discovered was that our process-modelling tool was very intuitive and easy to use. For an inexperienced user, it was possible to start to model a paper review process after a 5-minute introduction of the basic concepts. We also made the same user to model the same process with a more advanced textual role-based PML [17]. When using the latter PCE, it took several days before the user could do the modelling. In the role-based PML, it was possible to model much more advanced interaction between different roles and the visualisation of activities was more flexible. Our prototype produces simple web-based workflow support, where the activities are shown as web pages defined by the modeller. The few concepts in our PML makes it easier for inexperienced users, but also makes it

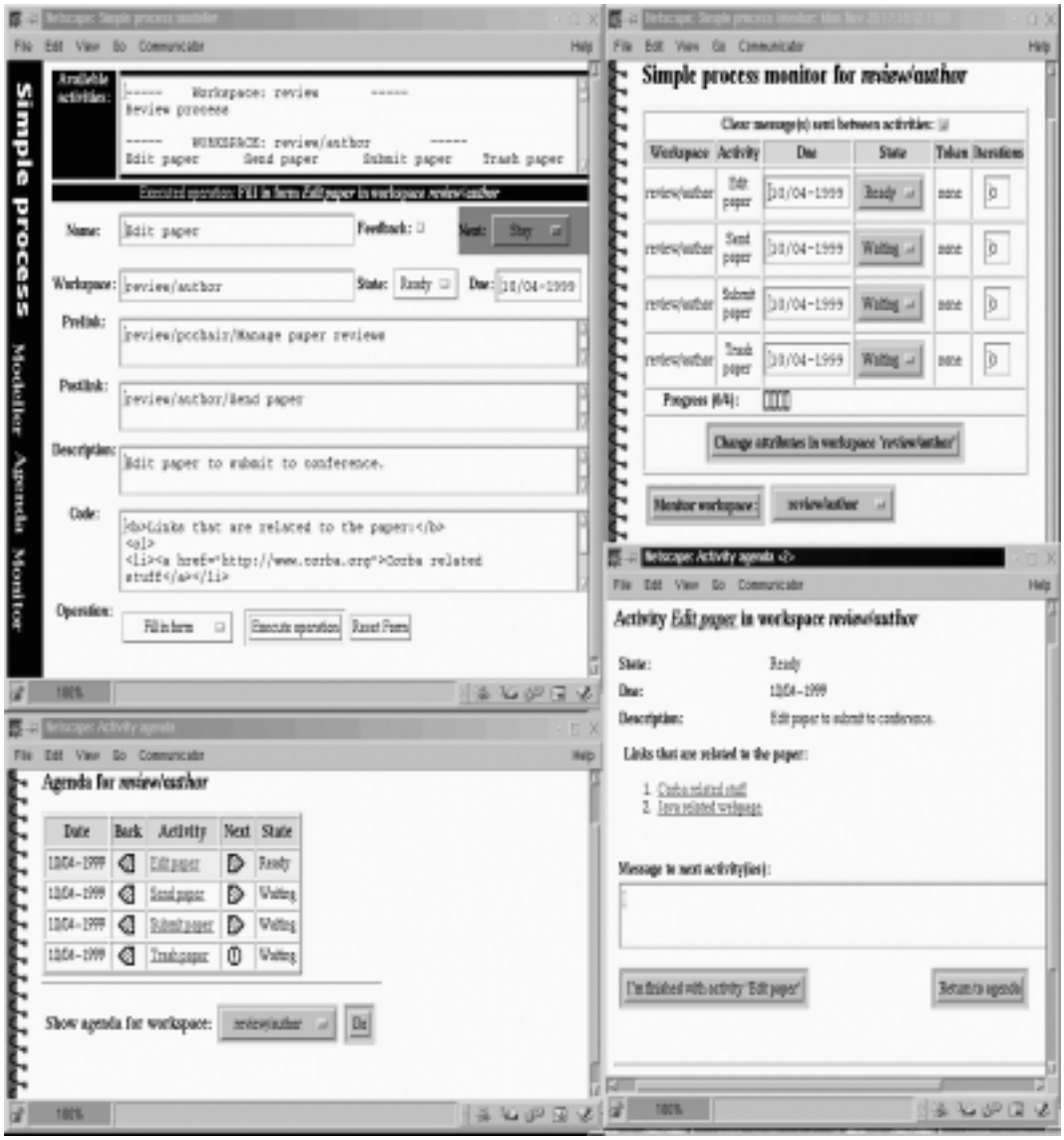

Fig. 8. Screenshots of the Process modeller tool, the Agenda manager tool, and the Monitor tool

more limited for advanced users. Since we want inexperienced users to interact, model and change their own processes, we chose the former approach.

One problem in letting process fragments move around is that users connected to workspaces can loose track of process fragments, and it is impossible to know where different process fragments have moved. We propose to solve this problems using software agents. In our multi-agent architecture (see section 1), we have identified monitor agents. Monitor agents are system agents used to monitor events in workspaces and Agent Meeting Places. Shared repositories are used to stored information gathered by the monitor agents. Repository agents are used to retrieve event information, and give answer to users working in workspaces. In this way, it is always possible to know where process fragments are located and what has happened to them by asking agents.

In the current version of our system, there are no restrictions on changing the activity definition (model) in a local workspace. Since an activity is defined by a simple XML-file, we propose to use a configuration management tool (CM-tool) to manage different versions of activities. Any CM-tool can be used since the XML-file is only simple text. The CM-tool can be used to track and manage changes of other files in the workspaces as well. As the system is today, it is up to the user to ensure that the changes result in a valid executable process model. This approach is quite similar to how web pages work on the Internet and it is possible to define dangling links between activities. This gives the users freedom to alter the process as much as they want, but also can generate a lot of problems. We have identified a need for a tool to help the users to the right choices when altering the process model. At least the tool should give a warning that certain changes can cause dangling links between activities. This means that we need a tool for analysing consistency and impact of process model changes.

In this paper we have proposed a framework for allowing a process model to be distributed on several places, and so that parts of the process can be moved around during enactment. In [16], a software development and maintenance process from the software process industry is described. A development department is doing all the coding and changing of code in this company. It is likely to think that a work-order or change-order issued to the development department also contains a process fragment. This means that a process fragment will be moved from the department issuing the work-/change-order to the development department. It is also possible to think of a scenario where a part of a software project cannot be done internally in the company. The process fragment defining this part of the process model can then for instance be moved and executed by an external consultant company. The external consultant company will get a process fragment along with the signed contract for the job. Outsourcing parts of software projects can then be supported in the PCE/workflow tool, even if the job is executed in another environment.

5 Conclusion

Our prototype has formed a basis for supporting mobile software processes within the CAGIS architecture. The integration of the whole architecture is still to be worked on,

to see the benefit from supporting mobile software processes. We expect to model and execute real-life CSE processes to gain useful experiences with moving software processes.

In this version of the prototype, there is no checking before adding an activity or a process fragment other than to check that activities already exist. This means that it is possible to instantiate process fragments that don't fit into the already running process model. We are currently working on a tool to check that process fragments fit into the model, before they are instantiated. Future work will try to solve this problem, and see how well real CSE processes can be supported in the CAGIS architecture.

6 Acknowledgement

We want to thank the Information Process Group, Computer science department, University of Manchester giving useful feedback on our work, and giving good introduction to the nice and flexible ProcessWeb environment. We would also like to thank Professor Chunnian Liu from Beijing Polytechnical University for commenting this paper and giving us useful input, and Professor Carlo Montangero and Professor Reidar Conradi for giving useful advises to improve the paper.

References

1. C. Basile, S. Calanna, E. Nitto, A. Fuggetta, and M. Gemo. Mechanisms and Policies for Federated PSEEs: Basic Concepts and Open Issues. In Carlo Montangero, editor, *Proceedings of the 5th European Workshop on Software Process Technology*, volume 1149 of *LNCS*, pages 86–91, Nancy, France, October 1996. Springer-Verlag.
2. Israel Ben-Shaul and Gail E. Kaiser. *A Paradigm for Decentralized Process Modeling*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1st edition, 1995. ISBN 0-7923-9631-6.
3. Richard Bentley, Thilo Horstman, and Jonathan Trevor. The World Wide Web as enabling technology for CSCW: The case of BSCW. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 7:21, 1997.
4. Supratik Bhattacharyya and Leon Osterweil. A Framework for Relocation in Mobile Process-Centered Software Development Environments. Technical report, Department of Computer Science, University of Massachusetts at Amherst, 23 August 1996.
5. Reidar Conradi, Marianne Hagaseth, Jens-Otto Larsen, and Minh Nguyen. EPOS: Object-Oriented and Cooperative Process Modelling. In *[6]*, pages 33–70, 1994.
6. Anthony Finkelstein, Jeff Kramer, and Bashar A. Nuseibeh, editors. *Software Process Modelling and Technology*. Advanced Software Development Series, Research Studies Press/ John Wiley & Sons, 1994. ISBN 0-86380-169-2, 362 p.
7. Jin Jing, Karen Huff, Himanshu Sinha, Ben Hurwitz, and Bill Robinson. Workflow and Application Adaptions in Mobile Environments. In *Second IEEE Workshop on Mobile Computer Systems and Applications*, New Orleans, Lousiana, USA, 25-26 February 1999.
8. Danny Lange and Mitsuru Oshima. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley, 1998.

9. Michael Merz, Boris Liberman, and Winfried Lamersdorf. Using Mobile Agents to Support Interorganizational Workflow-Management. *International Journal on Applied Artificial Intelligence*, 11(6), September 1997.
10. Geir Prestegård, Anders Aas Hanssen, Snorre Brandstadmoen, and Bård Smidsrød Nymoen. DIAS - Distributed Intelligent Agent System, April 1999. EPOS TR 359 (pre-diploma project thesis), 396 p. + CD, Dept. of Computer and Information Science, NTNU, Trondheim.
11. CAGIS project. Cooperative agents in global information space webpage. web: http://www.idi.ntnu.no/∼cagis, 2000.
12. J.W. Shepherdson, S.G. Thompson, and B.R. Odgers. Cross Organisational Workflow Coordinated by Software Agents. In *Workshop on Cross-Organisational Workflow Management and Co-ordination*, San Francisco, USA, February 1999.
13. Pierre F. Tiako. Modelling the Federation of Process Sensitive Engineering Environments: Basic Concepts and Perspectives. In Volker Gruhn, editor, *Proc. 6th European Workshop on Software Process Technologies*, pages 132–136, Weybridge, UK, 16-18 September 1998. Springer Verlag, LNCS 1487.
14. Alf Inge Wang. Experience paper: Using XML to implement a workflow tool. In *3rd Annual IASTED International Conference Software Engineering and Applications*, Scottsdale, Arizona, USA, 6-8 October 1999.
15. Alf Inge Wang, Reidar Conradi, and Chunnian Liu. Integrating Workflow with Interacting Agents to support Cooperative Software Engineering. In *Proc. IASTED Internation Conference Software Engineering and Applications*, Las Vegas, Nevada, USA, 6-9 November 2000.
16. Alf Inge Wang, Chunnian Liu, and Reidar Conradi. A Multi-Agent Architecture for Cooperative Software Engineering. In *Proc. The Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, pages 1–22, Kaiserslautern, Germany, 17-19 June 1999.
17. Benjamin Yeomans. Enhancing the world wide web. Technical report, Computer Science Dept., University of Manchester, 1996. Supervisor: Prof. Brian Warboys.
18. Jeong-Joon Yoo and Dong-Ik Lee. X-MAS: Mobile Agent Platform for Workflow Systems with Time Constraints. In *Proc. Fourth International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, 20-23 March 1999.