Using a Mobile, Agent-based Environment to support Cooperative Software Processes

THESIS

Presented in Partial Fulfillment of the Requirements for
the Degree of

Dr.ing.

By

Alf Inge Wang

Norwegian University for Science and Technology
Dept. of Computer and Information Science

February 5, 2001

TO MY WIFE AND CHILDREN,

and

IN MEMORY OF
MY MOTHER IN LAW,
NELLA GJÆRE,
WHO DIED OF CANCER
BEFORE THIS THESIS
WAS FINISHED.

Abstract

Cooperative Software Engineering (CSE) means that large-scale, software development and maintenance can be conducted in a distributed organisation or across organisations. CSE can be characterised by distributed process fragments, partly shared workspaces, co-operation planning, and frequent interactions in intra/inter-workspaces. To support CSE processes, we must deal with dynamic, unpredictable processes as well as stable, repeatable processes with totally different characteristics. Traditional workflow and process systems offer good support for stable, pre-planned processes, providing user agendas, invocation of tools, presentation of process state etc. Multi-agent systems are well suited to model and support users involved in cooperative processes. By combining these two technologies, processes with characteristics similar to cooperative software engineering processes can be modelled and supported more completely.

The thesis presents a framework called CAGIS Process Centred Environment (PCE), for combining a workflow system with a multi-agent system. These are the main parts of the thesis:

- A **multi-agent architecture** to support cooperative processes. This architecture is particularly useful in modelling and providing support for cooperative activities where software agents act on behalf of the user. The design and implementation of this architecture is described.

- A **workflow system** to support distributed mobile processes. This workflow system allows processes to be fragmented into smaller sub-processes that can be distributed over several workspaces and moved between these workspaces.

- A **gluing framework** to specify the interaction between the workflow system and the multi-agent architecture. The gluemodel defines the relationships between software agents and process fragments (sub-processes), and a GlueServer is used as a middleware between a workflow tool and a multi-agent system. Results from applying the GlueModel framework on a cooperative software engineering (CSE) process is also described.

- A **Evaluation of the framework** by modelling three practical cases:
  - A conference organising process is modelled in three different process environments (including our own), and evaluated according to modelling completeness and adaptability to process changes.
  - A CSE scenario describing a software and maintenance process in a Norwegian software company is modelled to show usefulness of the gluing framework.
  - A project organisation scenario used to demonstrate how software agents can be used in CAGIS Process Centred Environment to deal with evolution of distributed, fragmented workflow models.

Preface

This dissertation is submitted to the Norwegian University of Science and Technology (NTNU) in partial fullfillment of the requirements for the degree Doktor Ingeniør.

The work contained herein has been performed at the Department of Computer and Information Science, NTNU, Trondheim, under supervision of Professor Reidar Conradi, and at the Information Process Group, Computer Science Department, University of Manchester, UK, under supervision of Ian Robertson.

The work has been supported by grant 112567/43 from the Norwegian Research Council through the Distributed Information Systems (DITS) Programme

Thesis Structure

This thesis is an article collection where the main contribution is described through eight core papers. Further, the thesis contains a part setting the work in context and summarises what has been done. The thesis is organised into five parts as following:

Part I starts with an introduction chapter which describes the motivation and context for the thesis, and outlines the research questions. Chapter 2 describes state-of-the-art relevant to the thesis, whereas chapter 3 contains a description of various research methods that can be applied to evaluate technology in Software Engineering, and then describes the research focus and research method used in this thesis. In chapter 4, the thesis contribution is summarised, and chapter 5 evaluates our contribution. Future work is presented in chapter 6, and chapter 7 gives some concluding remarks.

Part II contains the *Background papers* that form the the knowledge and technology basis for the thesis.

Part III contains the core papers, and constitutes the main contribution of the thesis.

Part IV contains the appendixes for the thesis. Appendix A to C list the design specifications for the three main parts of the implemented prototype. Appendix D and E show screenshots from our evaluation of three process centred environments, while appendices F to H present the process models used to model a conference management process using three different process centred environments.

Part V lists all references for the whole thesis in a bibliography.

Acknowledgements

To write a thesis is a lot of work that is impossible to do completely alone. Many people have been directly or indirectly involved in my work, making it possible to finish this thesis.

First, I would like to thank my adviser Professor Reidar Conradi for the effort he has put into my work. Conradi has given me freedom to explore my research field, and helping me back on the right track when I have headed on in wrong direction.

Second, I would like to thank my family that have supported me and given me love all the way through my work. I would like to thank my wife Inger Synnøve Gjære for encouragement and support, and my daughters Stine Johanne Gjære Wang and Thea Camilla Gjære Wang for inspiration.

Further, I would like to thank Professor Chunnian Liu from Beijing Polytechnic University (BPU), and the CAGIS project group for useful feedback and suggestions for my work. I will especially thank these persons involved in the CAGIS project: Heri Ramampiaro, Terje Brasethvik, Sobah Abbas Pettersen, Monica Divitini, and Jens-Otto Larsen.

I would like to thank for the valuable six months I spent in Manchester where I was a part of the Information Process Group at the University of Manchester. I would especially like to thank Ian Robertson, and Mark Greenwood for interesting discussions and for giving me insight into the ProcessWeb workflow tool. I would also like to thank Pete and Share Hammond for being friends in Manchester, and to the Emmanuel Church, Didsbury for feeding my soul.

I would also like to give special thanks to my father Noralf Wang for proof-reading this thesis.

Through the work with this thesis, student projects have been used to implement prototypes and framework. I would like to thank these students for making contributions to my work: Håkon Rydland, Joar Øyen, Morten Simonsen, Christian Thuv, Geir Prestegård, Anders Aas Hanssen, Snorre Brandstadmoen, Bård Smidsrød Nymoen, Bjørn Haakenstad, Terje Salvesen, and Jan Waage.

I would also like to thank the software engineering group and the rest of the department for providing a nice working environment, and for useful discussions. Especially I would like to thank: Letizia Jaccheri, Minh Nguyen, Torgeir Dingsøyr, Øystein Nytrø, Guttorm Sindre, Roxana Elena Diaconescu, Bjarte Østvold, Hallvard Trætteberg, Amund Tveit, Babak Farshchian, Tom Reidar Henriksen, and Harald Haibo Xiao.

Further, I would like to thank the secretaries at our department that have always been helpful for doing various tasks. I would especially thank Lisbeth Vågan and Anne Berit Dahl for their kindness.

I have been privileged for the support from family and friends all the way through this thesis. I would like to thank my family Noralf and Bergny Wang, Nella and Svein Gjære, Karin, Oddgeir, Mats and Mari Holmen, John Arne Wang and Lisa Abrahamson, Anita, Per Ove and Andreas Børseth, Julia og Kyrre Knudsen, Siri Gjære and Steinar Raknes. In addition I would like to thank my closest friends with families: Thor-Ivar Dahlbakken, Torbjørn Slørdahl, Arnvid Hellebust, and Oddleiv Moen.

At last, I would like to thank my God for creating me,
and giving me ability to think and be creative.

Contents

List of Tables

List of Figures

Part I

Context

CHAPTER 1

Introduction

This chapter is an introduction outlining the context and motivation for the thesis.

1.1 Motivation

Today, software is a vital part of daily life e.g., in banking, insurance, construction of buildings, shopping, telecom, entertainment, cars, busses, trains, air-traffic etc. We are surrounded by all kinds of electronic equipment running some kind of software, from the smallest watches to large banking systems. It is estimated that about 6 % of Gross National Product (GNP) of industrial nations is spent on software (50/50 on COTS and tailor-made) in OECD countries. In the telecom business 70-80 % of the costs is related to software. Despite the fact that almost everything is dependent on software, the software industry is struggling with project delays, budget overrun, poor software quality, and software that is hard to maintain. One way of attacking the problems in the software industry, is to improve the process of producing software systems (development and maintainability).

From the early beginning of the software engineering era, software development processes have been modelled to understand them, to guide people involved in the process, to partially automate the process, and to improve the process. Inspired from production-line industry processes (e.g. in mechanical engineering), software development processes have been modelled as static and standardised processes. Traditionally, modelling and enactment of software processes have been focusing on ”forcing” and guiding people to work according to a specified model, where interaction between people has been coordinated through a strictly defined control/data flow. Cooperative aspects of the software development process have often been either eliminated or ignored, because it has been hard to model cooperative activities in existing systems, or there has not being an interest for doing so. Also, software development processes are human-centred processes. In [CG98], Cugola and Ghezzi state that ”Human-centred processes are characterised by two crucial aspects that were largely ignored by most software process research: They must support cooperation among people, and they must be highly flexible”. This thesis addresses these two challenges, and proposes a highly flexible framework for supporting cooperative software engineering processes. Evaluation of how existing software process technology (SPT) can cope with cooperative aspects has not been extensively performed. This is also a lack of validation of SPT in general. This thesis has investigated how cooperative activities can be supported in SPT.

In the workflow and Computer-Supported Cooperative Work (CSCW) community, some work has resulted in the development of cooperative workflow systems. Most of these systems are role-based systems, where the roles and the cooperative interaction between these roles are modelled. Lately, software agents have been used to model and enact cooperative activities. The software agents represent users in cooperative efforts and act according to the users’ requirements to reach a specified goal. By using software agents, we can benefit from the agents’ ability to learn and adopt to a changing environment. Activity-based workflow and process systems on the other hand, are efficient to model pre-planned activities that e.g., can be derived from a project planning tool. Activity-based workflow is not suitable for modelling cooperation, because interaction between roles are hard to represent in activity networks. Many workflow systems have also a problem to represent and support dynamic processes. This thesis presents a framework to combine software agents with activity-based workflow, to gain flexibility and to be able to model most aspects of a process.

1.2 Research Context: The CAGIS Project

This work has been executed as a part of a project called Cooperative Agents in a Global Information Space (CAGIS) that started in January 1997. The CAGIS project is a multidisciplinary research project supported by the Norwegian Research Council where three different research groups at the department of computer and information science have worked together. The CAGIS project team consists of people from a database group, an information system group, and a software engineering group.

The initial objectives for the CAGIS project [C$^+$96] were:

- To give cooperating human problem-solvers (designers, engineers) better support for concurrent and distributed team work.
- To provide a framework for the corresponding IT support, with distributed software agents and data stores, and with domain specific formalisms and tools.

- To provide for specialised software agents to operate in the global information space, e.g., Internet search agents and intelligent reconfiguration agents.

The work with the CAGIS project has resulted in a number of internationally published papers, several student projects, new courses within distributed technology and software agents, as well as a CAGIS environment (including prototype) [pro00] consisting of three main parts:

1.2.1 CAGIS Document Model Toolset

The CAGIS document model toolset helps users to semantically classify and describe documents published on the web, to make it easier to find and use these documents later. This work is done semi-automatically by using a conceptual modelling language (the Referent Modelling Language) to express a domain model, and by using text analysis tools to perform classification and search. The conceptual model is then used as a basis for creating meta-data descriptions of documents, that can be browsed or searched using a web/Java-based model viewer.

Fundamental to the CAGIS document model approach is the use of a conceptual modelling language to define and visualise the domain specific vocabulary to be used in the classification and retrieval process. Conceptual modelling languages contain the formal basis necessary to define a proper ontology, yet at the same time they offer a visual representation that allows users to take part in the modelling. In addition, we can use such languages to read and explore documents by interacting directly with the models. The conceptual modelling language may thus be used throughout the entire process of classifying and retrieving documents on the web. In this approach, we use the Referent model language [Arn98] being an ER-like language with strong abstraction mechanisms and sound formal basis.

A more detailed presentation of the CAGIS document model approach is given in [Ter99, Ter00].

1.2.2 CAGIS Transaction Specification Framework

The CAGIS transaction specification framework is a transaction framework used to provide configurable, application-specific transaction models [RN00]. By making it possible to adjust the degree of control provided by transaction models, the framework can support situations demanding strict control for data correctness as well as situations with more relaxed (i.e. user-define) correctness and operational rules. The framework consists of two parts:

- *Transaction characteristics specification* defining the main properties of the actual transaction: degree of ACID properties, relationships among the involved transactions to be executed, adopted correctness criteria, and applied policy. These characteristics are statically defined and must be defined before the designated transactions are executed.

- *Transaction execution specification* defines how the transaction execution is to be performed at run-time, in terms of composition of management operations (e.g., delegate, abort, write, etc.) and regular access operations (read, write etc.).

Within the CAGIS framework the CAGIS transaction specification framework can be used to manage the execution of software agents, by letting the CAGIS transaction manager control the execution of agents. If an agent fails, the system can roll-back to a safe-state. In addition, the CAGIS transaction manager can manage evolving workflow models, by specifying how workflow models can be changed in terms of transactions.

The transaction management system described above was implemented in a working prototype [Sel00, KK00] based on Java and the IBM Aglets-workbench. It has served as a test-bed for the transaction specification framework.

1.2.3 CAGIS Process Centred Environment

The CAGIS Process Centred Environment (PCE) provides process support for people that are distributed, working with cooperative activities as well as individual activities. The CAGIS PCE provides a framework to support evolving processes, and to allow autonomous parts of the process (process fragments) to interact. Simple individual activities are modelled and enacted by the CAGIS SimpleProcess workflow tool [Wan00b]. Cooperative activities and interaction between autonomous process fragments are supported through a multi-agent architecture, called CAGIS DIAS [WLC99, Alf00, Wan00a], with cooperation agents. A so-called *GlueServer* is used as a middleware between the CAGIS SimpleProcess workflow tool and the multi-agent architecture, using a GlueModel to define the interaction between process fragments and interacting agents [WCL00]. The work presented in this thesis focus on the CAGIS PCE.

1.2.4 The CAGIS Environment

The CAGIS environment consists of a set of separate tools that may be used together to provide support for cooperative work across the web. The three major components of the CAGIS Environment are the above mentioned CAGIS Document model toolset, CAGIS Transaction manager, and CAGIS PCE. Each of these tools is implemented in true Web style, i.e. they are built around a standard Web server and use XML as a data storage and interchange format. These tools may all be configured according to the actual situations and usage. Central to the CAGIS environment, and binding the individual tools together, is the CAGIS GlueServer. The GlueServer, configures a set of software agents that can activate the different CAGIS tools. The *GlueModel* defines the relationships between the individual workflow elements that may reside in different workspaces and the software agent that may be used to access the individual tools. In this way, the various components of the CAGIS toolset may be used together in order to provide situation specific cooperative support.

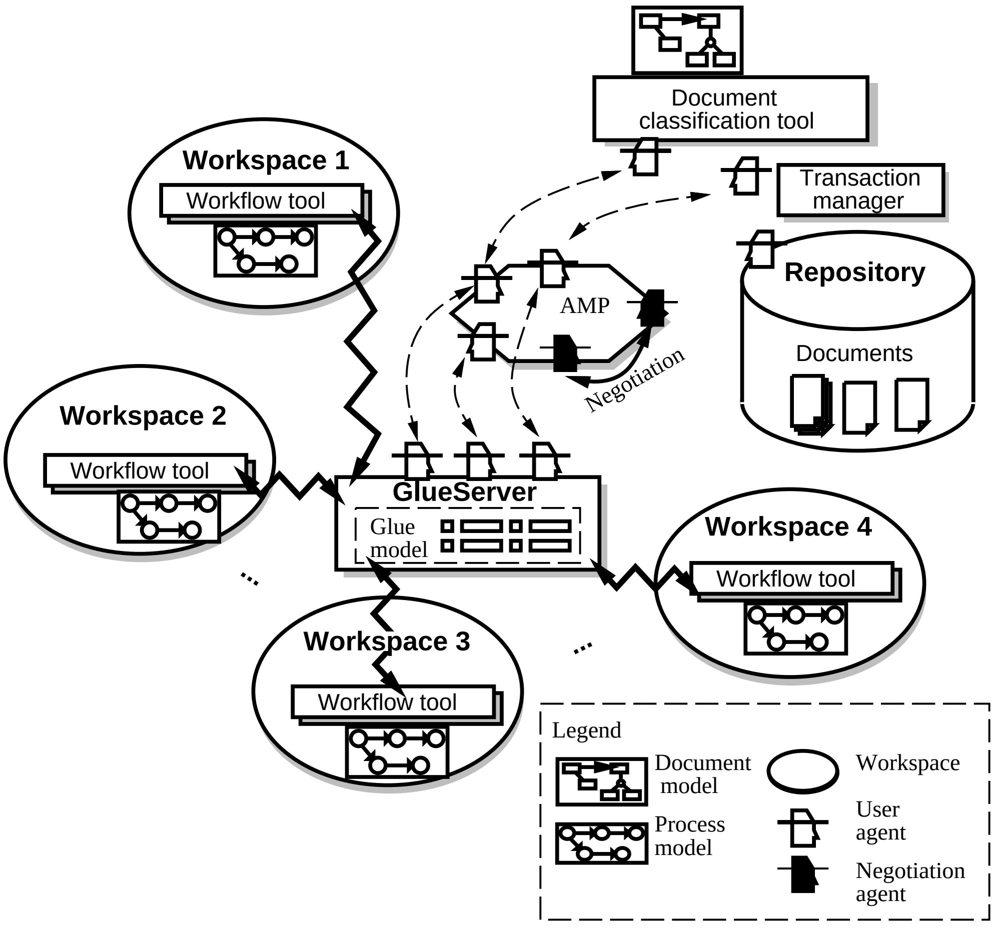


Figure 1.1: The CAGIS Environment

Figure 1.1 illustrates how the different parts of the CAGIS environment interacts. In each workspace where a user or a group of users works, the CAGIS SimpleProcess workflow tool is used to enact the individual processes. The cooperative activities and interactions between workspaces are supported through the GlueServer. The workflow tools in the workspaces will notify the GlueServer about state changes, and the GlueServer will search through the GlueModel for workflow state matches. If a match is found, the GlueServer will initiate one or more software agents that provide cooperative services for users such as resource negotiation, coordination of artifacts, activating tools etc. A document agent can activate the document classification tool that can be used for building a domain model for shared documents. This domain model together with the document classification tool can also be used by users in the various workspaces through document agents for browsing documents, searching for specific documents or parts of the documents, classifying documents etc. The document classification tool helps users to share common knowledge and information which is also an important aspect of cooperation. The transaction manager is responsible for managing the integrity of the documents in the repository, and to ensure that agents always leave the system in a consistent state. There is a change for shared documents to be updated at the same time. To insure consistency, the transaction manager can use three main approaches. The first approach is to only permit an exclusive lock on documents, allowing only one person to access a document at a time. A second and a more relaxed approach is to allow reading access to a document while it is updated. The third approach is to permit simultaneous updates, where consistency can be achieved only through multiple-version handling with a sophisticated merging mechanism. It should be noted that software agents are not only activated by the GlueServer, but can also be directly be activated by the users from their workspaces.

In addition to use the various CAGIS tools together as one CAGIS environment, one CAGIS tool can be used to enhance functionality in another. n [Alf00], an example of how the document models and tools can enhance the CAGIS multi-agent architecture is given. The document models and tools are here used to model the agent ontology, which define the language software agents can speak. In [Wan00c], the transaction models and tools (in this paper called workspace manager) offer a way managing consistency of changing workflow models. This means that the CAGIS environment offers a selection of tools, which can be used in different combinations to give specific support.

A full description of the CAGIS environment applied to a conference organising scenario is presented in the paper ”*Supporting Distributed Cooperative Work in CAGIS*”, found in chapter 13.

1.3 Research Questions

One of the main goals in the CAGIS research project was to give humans working in a heterogeneous and distributed environment adequate support for process modelling and enactment. Using this as a starting-point, this thesis will address the following research questions:

1. **Modelling:** Investigate *what is needed* to model and enact distributed, cooperative and individual processes in a heterogeneous environment. This research topic can be further decomposed into:

   1a) Investigate what formalism is needed to model and enact individual processes (for individuals or groups).

   1b) Investigate what formalism is needed to model and enact distributed, cooperative processes.

   1c) Investigate how the process can be distributed among the participants.

   1d) Investigate how to model and support dynamic process changes.

2. **Tools:** Investigate *how to create* an infrastructure for providing process execution of distributed, cooperative processes in a heterogeneous environment.

   2a) Investigate what architectures are needed to provide execution support for individual processes.

   2b) Investigate what architectures are needed to provide execution support for distributed, cooperative processes.

   2c) Investigate what technology can be used to provide a distributed, heterogeneous execution environment.

   2d) Investigate how different technologies (from 2a, 2b and 2c) can be combined.

   2e) Investigate how to provide an open-ended process architecture allowing other process systems to be a part of the environment.

3. **Validation:** Investigate how the proposed approach compares to other approaches using a practical case.

Propositions to these research questions are described in chapter 5.2.

CHAPTER 2

State-of-the-art

This chapter gives an introduction to the research fields of software engineering, computer-supported cooperative work (CSCW), workflow, software agents, and middleware. The first part of this chapter (section 2.1) gives definitions of the central terms for the research fields, and discusses how these research fields are related. The rest of the chapter describes each research field in more detail (section 2.2-2.7), identifies some research challenges in these research fields, and describes how our approach relates to these research fields and challenges.

2.1 Definitions and Terminology

This section outlines definitions of central terms used for software engineering (with emphasis on software processes), CSCW, workflow, software agents, and middleware. The relationships between the research fields are also discussed. A more detailed description of the research fields is given in the sections 2.2 to 2.7.

2.1.1 Software Engineering: Software Process

Software engineering is according to Webopedia [1] defined as:

- “*The computer science discipline concerned with developing large applications. Software engineering covers not only the technical aspects of building software sys-*

[1]Webopedia is a web-based online dictionary that can be found at http://webopedia.internet.com

> *tems, but also management issues, such as directing programming teams, scheduling, and budgeting*” [Web00].

The definition above states that there is much more to software engineering than simply writing code. For development of rather small applications, the main focus will be on efficient methods for producing high quality code. Software engineering however focuses on producing large applications, and thus efficient management of resources like people, tools, knowledge is crucial for the outcome of projects. Sommerville [Som95] describes software engineering as:

- “*Software engineering is concerned with software systems which are built by teams rather than individual programmers, uses engineering principles in the development of these systems, and is made up of both technical and non-technical aspects*”.

Sommerville’s definition emphasises that software engineering is a collaborative activity rather than programming by individuals. This view of software engineering fits very well with the focus of this thesis which is on cooperative software development processes. Since the focus of this thesis is on software processes, the rest of this subsection will focus on related definitions.

The term *software process* [LB85] is used to denote all activities performed within a software organisation. *Software process modelling* describes the activity of understanding and describing a given software process. The term *Software Process Technology (SPT)* is used about the concepts, languages, methods and tools used to support software processes. According to Høydalsvik [Høy97] **SPT** can be defined as following:

- “*Methods and tools that aim to support the definition and execution of software development processes within a software development organisation (typical support includes modelling, analysis, and evolution)*”

From the descriptions and definitions of the software process related terms above, we can see that neither of them explicitly mention any cooperative aspects software development. Traditionally in software engineering, software development processes have been seen from a top-down perspective, where higher management has coordinated and controlled the software developers. In this thesis, we also want to add a perspective where the cooperative aspects of software development are taken into account as well. Creative processes such as software development are dependent on cooperative effort between teammembers.

In the introduction of the PROMOTER-2 book “Software Process: Principles, Methodology and Technology” [DBW98], the term SPT is used to describe “the integration of the production and the management technologies in a comprehensive work environment also called Process-sensitive Software Engineering Environment (PSEE) that supports the whole process”. This means that a PSEE can be seen as a realisation of SPT. Figure 2.1 shows the impact of this technology, and illustrates how the PSEE implements, controls and enhances both the feed-back and the feed-forward paths by which the management process controls the production process. A PSEE supports both the production and management process for software production. The production process will change due to changes in the working environment, while the management process standardises ad-hoc routines in the working environment. Further, the PSEE integrates production, process and management technology into an engineering environment used by people in the working environment.

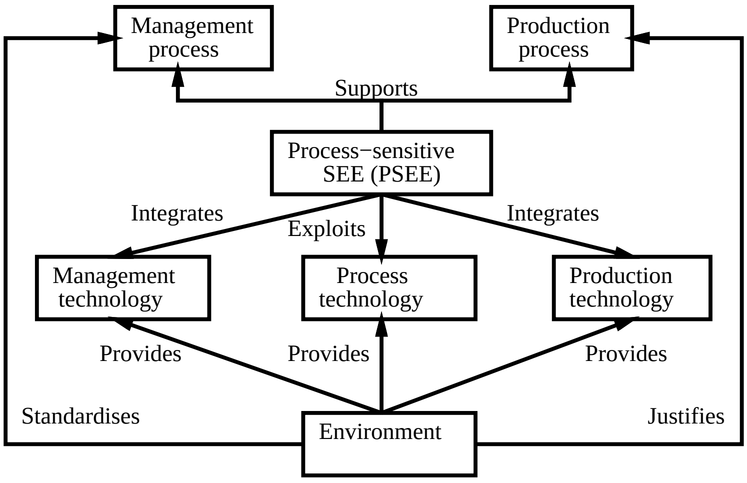


Figure 2.1: The impact of software process technology

The basic concepts in a **process model** are shown in figure 2.2 [DBW98]: An *activity* is an atomic or composite operation, or a step of a process. A *product* is a set of artifacts to be developed, delivered and maintained in a project. A *resource* is an asset needed by an activity for it to be carried out. There are two important resources: the *performers* (e.g. human agents) and the *tools*. Performers are usually indirectly connected to an activity via their *roles* that can be used to describe responsibilities, obligations and skills. The *directions* are policies, rules and procedures (i.e. process models) that govern the activities. Figure 2.2 shows how the different concepts in a process model are related.

To be able to offer process support, the software process must be modelled. **Software process modelling** can according to [Høy97] be defined as:

- "*The activities performed, and language and tools applied, in order to create models of software development processes within a software development organisation*".

This definition is rather loose and does not say anything about what activities are typical when modelling a process, and what process elements should be included in a process model as described in figure 2.2. Activities that might be regarded as a part of a software

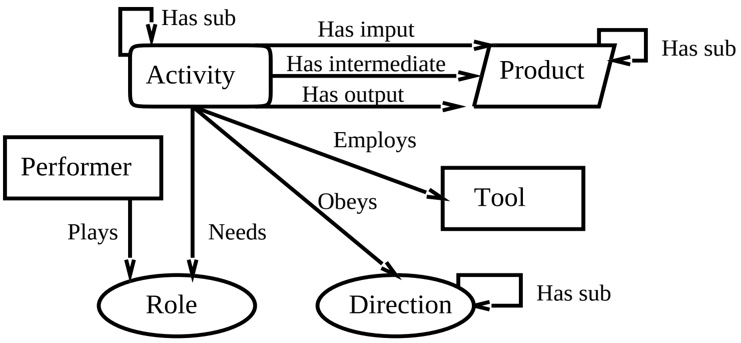


Figure 2.2: Basic concepts of process modelling

modelling process are: Assessing process knowledge, identifying process elements (activities, products, roles etc.), detailing process element descriptions, and relating process elements.

Software process models can be used to understanding the process better through graphical notations, but also to give active process support for the process participants. The execution of software process models is called **software process enactment** and can be defined as [Høy97]:

- "*The activities performed, and languages and tools applied, in order to create and use process programs which support software development processes within a software development organisation*".

This definition has similarities to the definition of software process modelling. The main difference is that the goal of software process enactment is to execute process models (here named process programs) to support the software development.

2.1.2 Workflow

The Workflow Management Coalition is an organisation that has managed to standardise workflow management through defining workflow terminology, defining a workflow architecture, and by defining standard workflow application programming interface (API). **Workflow** is according to the Workflow Management Coalition defined as:

- "*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*" [WfM99].

From this definition we can conclude that the goal of workflow is to execute business processes more efficiently, by coordinating activities or information.

The term process definition is used within workflow to denote the model of the process. The Workflow Management Coalition has defined **Process Definition** as:

- "*The representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.*" [WfM99].

How does Workflow relate to the other Domains?

When comparing the definitions for SPT and workflow, we can assume that the main difference is that the former focuses on software engineering processes, while the latter focuses on business processes. The definition of SPT is more general and includes evolution of the process as part of it. General business processes are often stable and can also often be predefined, since they consist of routine tasks such as routing documents for approval etc. Software processes have also some management tasks that are the same for every project, but much of the work is creative activities. In addition, software processes change frequently during projects because the customer issues new or revised requirements, project managers tend to under-estimate various tasks under planning, the customer delays to deliver documents, software developers are re-allocated to other projects and so on [NWC97].

Compared to to the definition of *Software Process Modelling* (SPT), the definition of *Process Definition* (Workflow) is more detailed, describing what information a workflow model should contain. From the definitions above, the main difference between research in the workflow community and the SPT community has been on what domain the process models are applied to and focus on process evolution. This might indicate that SPT should be able to handle creative human-centred work like design work, that requires dynamic process changes being initiated by the persons involved. This view indicates that SPT should be *regarded as CSCW*. The term *Process Centred Environment* (PCE) will in this thesis be used to categorise both workflow systems, PSEEs, and CSCW systems with process support.

2.1.3 Computer-Supported Cooperative Work (CSCW)

Although there has been a huge interest in the CSCW research-field, there is no universally accepted definition of CSCW [Wil91]. According to Bannon [Ban93], there is at least five distinct ways of viewing CSCW:

1. **CSCW as simply a loose agglomeration of cooperating and at times competing communities.** This view sees CSCW as an umbrella term with little content other

than something that has to do with people, computers and cooperation. Therefore CSCW can be seen as a forum where people from different disciplines and with partially overlapping concerns can discuss issues of mutual interests.

2. **CSCW as a paradigm shift.** Here CSCW is regarded as a paradigm shift because the organisational and human factors are taken into account when designing computer support systems, as opposed to technology-focused design of computer support systems.

3. **CSCW as software for groups.** This view sees CSCW as a research field focusing on people working with computers in groups. The term *groupware* is often used to name this area of CSCW.

4. **CSCW as technological support of cooperative work forms.** Here the emphasis is on understanding cooperative work as a distinctive form of work, and on supporting these cooperative work forms with appropriate technology.

5. **CSCW as Participative Design.** This view on CSCW is an alternative to traditional system design, where users are involved more thoroughly in the design process.

To summarise the five views above, we can say that CSCW focuses on human-centred computing with emphasis on cooperation. In this thesis, CSCW will be regarded as software for groups (view 3) with focus on cooperation between groups and between individuals in a group.

How does CSCW relate to the other Domains?

Workflow is often seen as a part of the CSCW research, but has also strong relations to office automation. According to Bannon's five views of CSCW in section 2.1.3, workflow can be categorised under CSCW as software for groups (view three), and CSCW as technological support of cooperative work forms (view four). SPT research has traditionally been quite opposite to CSCW, because SPT has been focusing on improving software processes from a top-down point of view. SPT has helped managers to improve and automate the software engineering without giving the same support for the developers doing the actual work. This is also the case for some research within workflow. In CSCW, the focus has been on the people doing the actual work, and how to support these people to achieve efficient cooperative work. More recent research within workflow and SPT has been focusing more on the CSCW aspects.

2.1.4 Software Agents

As with CSCW, there is no consensus on how to define the term software agents. Bjørn Hermans has suggested two definitions that can be used on software agents [Her97]:

1. "*An agent is a software thing that knows how to do things that you could probably do yourself if you had the time.*" Ted Selker of the IBM Almaden Research Center (quote taken from Janca (1995) [Jan95])

2. "*A piece of software which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. The software should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result.*" (informal definition by G. W. Lecky-Thompson)

The two definitions above are informal and rather vague. They both claim that software agents can act on its own (autonomy). In addition, the second definition emphasises that an agent should take the surroundings and changes in the surroundings into account when deciding what to do next. This makes the second definition better than the first. Another way of defining agents is to look at some characteristics that agents should hold [Her97]:

- **The Weak notion of the concept "agent".** Perhaps the most general way in which the term agent is used, is to denote a hardware or (more usually) software-based computer system that has the following properties:
  Autonomy (agents operate without the direct intervention of humans or others), *Social ability* (agents interact with other agents and (possibly) humans), *Reactivity* (agents perceive their environment, and respond in a timely fashion to changes that occur in it), *Proactivity* (agents are able to exhibit goal-directed behaviour by taking the initiative), *Temporal continuity* (agents are continuously running processes not once-only computations or scripts that map a single input to a single output and then terminate), and *Goal orientedness* (an agent is capable of handling complex, high-level tasks).

- **The Strong(er) notion of the concept "agent".** For some researchers, especially in the AI research field, the term agent has a stronger and more specific meaning. The following characteristics could be used to define such agents:
  Mobility; ability to move around in an network, *Benevolence*; assumption that agents do not have conflicting goals, *Rationality*; assumption that an agent will indeed act in order to achieve its goals, *Adaptivity*; (should be) able to adjust itself to the habits of its user, and *Collaboration*; an agent should not unthinkingly accept instructions, but should take into account that the human user makes errors and omits important information.

- **"Agency" and "Intelligence".** These characteristics denote that agents must have a certain degree of autonomy and authority, and should have intelligence. A short definition of intelligence could be: *"Intelligence is the degree of reasoning and learned behaviour...*" [Apa95].

- **The User's "Definition" of Agents.** Researchers often define what a software agent is. However, it is important that the users of agents also contribute to define software agents. Users will not start to use software agents because they are e.g. proactive or reactive, but rather because they can help them to for instance make work easier.

As we can see from the descriptions above, the definition of a software agent is a bit illusive. We have therefore suggested our own definition of an **agent**:

- "*A piece of autonomous software created by and acting on behalf of a user (or some other agent). It is set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to environment, as well as pro-activeness.*" [WLC99]

How do Agents relate to other the Domains?

We can see some similarities between software agents and CSCW. Characteristics like social abilities, reactivity, and collaboration fit very well into the CSCW world. From a CSCW point of view, software agents can be viewed as an enabling technology that can be used to provide cooperative support. Software agents are however also used to implement workflow systems and PSEEs.

2.1.5 Middleware

Middleware is a term that is used to describe some software that can serve as the glue between applications. According to Webopedia middleware is defined as:

- "*Software that connects two otherwise separate applications*" [ISP00b].

The definition of middleware above can be a bit limited depending on what the term *software* includes. In this thesis, an extended definition of middleware is defined as:

- "*Software that connects two otherwise separate applications and models describing this connection.*"

How does Middleware relate to the other Domains?

Middleware is a research area creating technology that can be used across the research fields SPT, workflow and CSCW. Middleware can be used to glue components within SPT, workflow, and CSCW systems separately, or as bridges between technologies from these research fields. In software engineering, PSEEs use middleware e.g. to create federated environments consisting of tools and systems with various interfaces and APIs, running on different platforms.

2.1.6 Summary

To summarise this section, we have compared the five research fields according to the domains they are applied to and the goals they try to achieve as shown in table 2.1.

Research field	**Domain(s)**	**Goal(s)**
SPT	Software development processes	Improve software development processes (efficiency, understanding, management, process evolution).
Workflow	Business processes	Automate (conform) business processes.
CSCW	Administrative work, engineering, education games etc.	Improve cooperative user support.
Agents	Computer games, information retrieval, information filtering, industrial process control, user interface design etc.	Intelligent systems, Efficient distributed computing, Decomposition of complex tasks, Evolving systems.
Middleware	Distributed systems, object technologies, architecture, security, software engineering, etc.	Connecting separate systems and components.

Table 2.1: A small comparison of five research fields

The table shows that both the SPT and the workflow fields are domain specific, while the rest can be applied to several domains. This is a simplification because SPT and workflow can be applied to other domains as well, but are usually applied to software engineering and business processes respectively. The goals of the research fields also reflect this same tendency. For instance, one of the main goals for the CSCW community has been to improve cooperative user support (with little enforcement). Cooperative computer systems can be used in various domains such as in software engineering, in business processes, in collaborative authoring, in collaborative graphical design etc. Middleware has also a goal that is applicable on various domains, namely to connect separate systems and components. The agent field is the only research area with several separate and possible colliding goals. For instance it can be very hard to achieve an intelligent system that computes efficiently over a network through mobile agents. Intelligent agent systems often require very large programs which are not very well suited to be moved between nodes in a network as mobile agents.

2.2 Software Engineering

This section describes the research field software engineering, with emphasis on software processes since these are the focus of this thesis.

2.2.1 Introduction

In 1968 at a NATO Conference in Gramisch-Partenkirchen [NR69], the term *Software Engineering* was introduced. The conference discussed what then was called the "*software crisis*" introduced by third generation computer hardware. The third generation computers were several times more powerful than previous hardware, and made it possible to run and implement large software systems. Some initial experiments showed that existing software development methods could not be scaled up to manage large software projects. As a result most large software projects were delayed, costs were more than estimated, were less reliable, had a bad performance, and were hard to maintain. Since the late sixties, the computer hardware has become less expensive and more powerful, enabling even larger software systems to be built. As mentioned in the introduction of this thesis (section 1.1) software has become a very important part of everyday life. Despite the fact that almost everything is dependent on software, the software industry is still struggling with the same "software crisis" as they did in the sixties.

From the definitions of software engineering in section 2.1.1, it is clear that software engineering involves everything that can impact on the how we develop software. Software Engineering typically includes concepts, theories, paradigms, languages, methods, and tools. Figure 2.3 shows an overview of software engineering, divided in two. The upper part shows vertical software engineering techniques for each development phase in the software process. The lower part shows horizontal techniques that are independent of the development phases. Note that there exist more techniques than shown in the figure such as software engineering environments, software engineering databases etc.

In [Bro86], Brooks argues that there is no "silver bullet" that can solve all the problems in software engineering. There is no single development, technology, or management technique that solve problems with modern software systems as complexity, conformity, changeability and invisibility. Brooks uses the term invisibility to denote that the software structure is invisible for the user, and that it is hard to visualise software.

The rest of this section will describe different research areas within software engineering, starting with the software process.

2.2.2 The Software Process

The word *software process* was coined by Manny M. Lehman in 1969 when he spent a year studying an IBM programming process to propose new research projects to improve

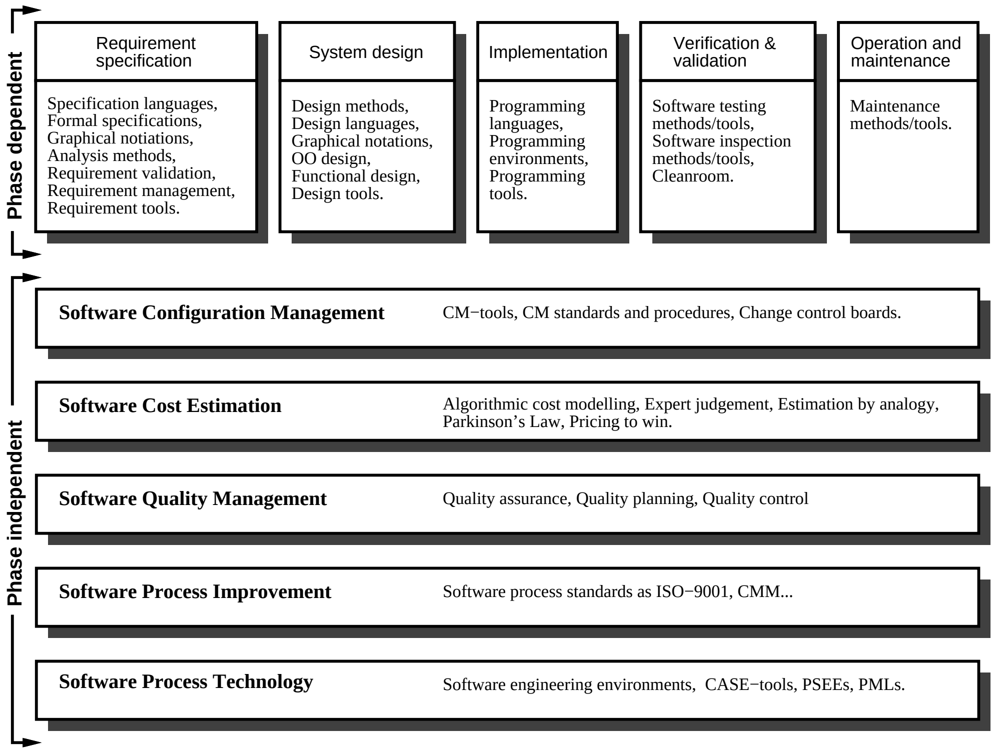

Phase dependent
Requirement specification
Specification languages, Formal specifications, Graphical notiations, Analysis methods, Requirement validation, Requirement management, Requirement tools.
System design
Design methods, Design languages, Graphical notations, OO design, Functional design, Design tools.
Implementation
Programming languages, Programming environments, Programming tools.
Verification & validation
Software testing methods/tools, Software inspection methods/tools, Cleanroom.
Operation and maintenance
Maintenance methods/tools.
Phase independent
Software Configuration Management
CM−tools, CM standards and procedures, Change control boards.
Software Cost Estimation
Algorithmic cost modelling, Expert judgement, Estimation by analogy, Parkinson's Law, Pricing to win.
Software Quality Management
Quality assurance, Quality planning, Quality control
Software Process Improvement
Software process standards as ISO−9001, CMM...
Software Process Technology
Software engineering environments, CASE−tools, PSEEs, PMLs.

Figure 2.3: An overview of Software Engineering

IBM's programming capabilities. In [Leh69], the software process was named *programming process* and was defined as "*the total collection of technologies and activities that transform the germ of an idea into a binary program tape*". Lehman's acknowledgement of the "programming process" initiated the research field software process.

Overall Views on the Software Process (Lifecycle Models)

According to Cugola and Ghezzi in [CG98], the view of the software process has evolved from viewing the process as a black box to viewing the process transparently. When viewing the software development process as a *black box*, the product (output) only depends on the product requirements (input). In this view, the process of producing a product from the product requirements is not defined at all. The problem of using this model, is that it is hard to predict when and with what costs the product will be finished. Often, the product requirements are informal and uncertain when starting a software development project. This means that it is likely that the output (product) of the process is not what the user wants.

To reduce the risks of failed software development projects, it is important to open the black box and have a more detailed look at the process. The *software life-cycle* and *Waterfall model* [Roy70] were defined to describe the life of a product from initial stage to final production. The software development process was refined from being a black box to a predefined sequence of phases, where each phase has a set of defined activities. Each phase has an input and output, and the output from one phase is the input to the next. The Waterfall process is in principle linear and a succeeding phase should not start until the previous phase was finished. The motivation for this was that the developers should think through everything in advance before the next phase, to avoid a sloppy attitude towards controlled software development. However, in practice phases overlap making it hard to follow the Waterfall model as defined, e.g. if problems with requirements are discovered in design.

The *evolutionary development model* is another way viewing of the software development process. This approach interleaves the phases specification, implementation and validation, and is based on rapid prototyping. A prototype is developed early in the project based on some initial requirements. This prototype is then refined through interactions with the customer until the customer is satisfied. The *Spiral Model* [Boe88] is an example of an evolutionary development model that can incorporate different development processes through separate iterations. There are two main types of evolutionary development: *Exploratory development* where the objective is to work with the customer to explore their requirements and deliver a final system, and *Throw-away prototyping* where rapid prototyping is only used to understand the customers' requirements. Evolutionary development is also often called incremental development, since a software system is built through several software increments rather than one final delivery.

Figure 2.4 illustrates the differences described above between the waterfall development model and the evolutionary development model.

Software Process Phases

A software process is usually decomposed into some phases with associated software engineering methods attached to them. The Waterfall model identifies the following five phases:

1. **Requirement specification** is the process of specifying the customer requirements of the final software system to be developed, and is usually divided into functional, non-functional, and domain requirements. Requirements are usually specified in natural language, but there are supplementary approaches to enhance the quality of the requirement specification. *Structured natural languages* are defined standard forms or templates to express the requirements specification. *Design description languages* use a programming language with more abstract features to specify the requirements by defining an observational model of the system. A Program Description Language (PDL) derived from programming languages like Java or Ada can be used for this purpose. *Mathematical specifications* use mathematical notations such as finite-state machines or sets to specify requirements resulting in a

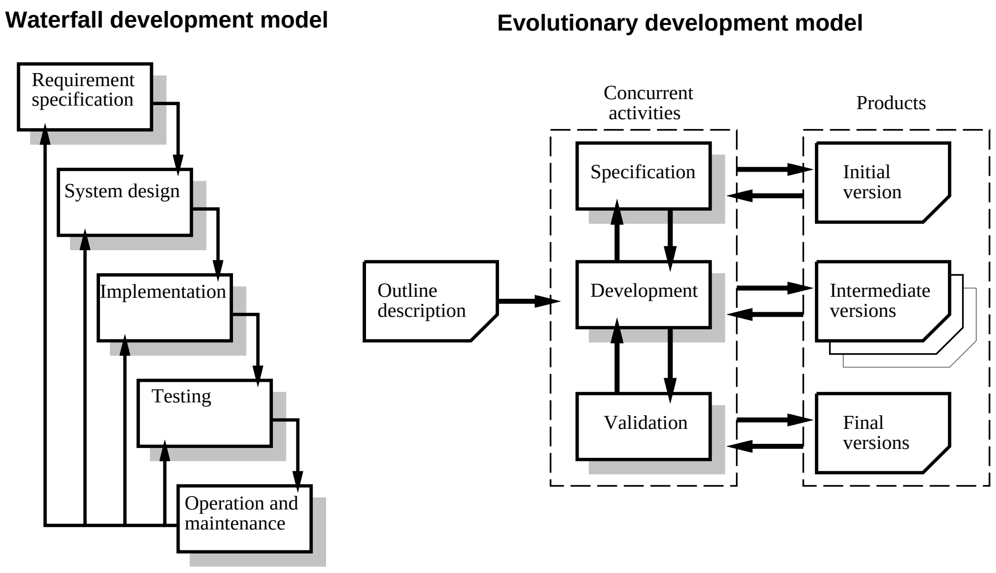


Figure 2.4: Waterfall development model vs. evolutionary development model

formal specification. The main problem with this approach is that most customers don't understand such specifications. *Graphical notations* describe requirements through graphical symbols, texts and arcs. In UML, the *use-case model* is used for this purpose.
In addition to specification languages, methods and tools for problem analysis (e.g. Viewpoint-Oriented Requirements Definition (VORD) [KS97]), requirements validation (e.g. prototyping), and requirement management (e.g. requirement change management) are used to aid this phase.

2. **System design** is the process of describing how the requirements are going to be implemented, and includes identifying sub-systems, establishing the software architecture, and describing how to create all sub-systems. The design process is a creative process, and often software design is an *ad hoc* process. However, there are structured design methods that can be used when designing software consisting of sets of notations (often graphical) and guidelines about how to create a software design. There are two main design strategies: *Functional design* where the system is designed from a functional viewpoint (e.g., Structured Design [YC75]), and *Object-oriented design* where the system is viewed as a collection of objects (e.g. Object-Oriented Development (OOD) [Boo91], Hierarchical Object-Oriented Design (HOOD) [HM93], Object Modelling Technique (OMT) [RBP$^+$91], Responsibility-Driven Design (RDD) / Class-Responsibility-Collaboration (CRC) [WBWW90], and Object-Oriented Analysis (OOA) [CY90]). Many graphical notations proposed for object-oriented design is now a part of the Unified Modelling Language (UML) [Cor00b] that has become the industry-standard language

for specifying, visualising, constructing, and documenting the artifacts of software systems. Note that UML is also used for other phases of software development.

3. **Implementation** is the process of realising the software design as a set of executional programs. Programming languages have evolved from being simple hardware-based binary code, through mathematical based languages like FORTRAN, to high-level, hardware-independent object-oriented languages like Java. Today we have a variety of programming languages for different purposes like Prolog (AI), AWK and Perl (text processing), ADA (military), Simula (simulation), and Java scripts (Web-browser programming). In addition to the programming languages, the programming environments have always been an important part of software engineering. Environments like Gandalf [HN86] and Cedar [SZBB86] tried to ease the programming process by structuring the programming through a programming language, selective user interfaces, tools, a common development database and integration of these elements.
   Modern programming environments like Visual Studio from Microsoft [Cor00a], offers a broad spectrum of tools for software development, including user interface drawing tools, reusable component browsing, software configuration management, task management etc.

4. **Testing** is the process of checking if the software conforms to its specification and meets the need of the customers. The most common techniques for testing are *software inspections* and *software testing*. Software inspections analyse and check system representations such as requirements document, design diagrams, and the program source code. Software testing involves execution of the final system with test data and examining the outputs and the operational behaviour of the system. There are several types of software testing, such as unit testing (white box), integration testing, function testing (black box), regression testing, system test, and acceptance and installation tests. Cleanroom [MDL87] is a software development method used for avoiding software defects through a defined inspection process. The system is formally specified, the formal verifications are used to verify the code. Testing can also be applied in phase 1-3.

5. **Operation and maintenance** is the process of installing the system, putting the system in operation and correcting errors that have not been discovered before this phase. In many software project, this is the longest phase.

Customisable Process Support by Process Modelling

The term ”Process Programming” descends from Leon Osterweil’s keynote speech at the 9th International Conference on Software Engineering. The title of Osterweil’s speech was ”Software processes are software too” [Ost87] and was the beginning of research on explicit software process modelling and enactment, and on customisable software processes. According to Osterweil, software organisations vary and their processes varies. It is therefore necessary for environments that support software development is able to adapt to special needs. To be able to explicitly support tailorable software development processes, process modelling languages (PMLs) were needed. The process models should also be executable, to provide runtime process support. Lehman in [Leh87] criticised Osterweil's keynote and said that in reality it is very hard to model software processes in such details that they can be executable, since software processes evolve and changes over time and consist of creative tasks. Thus, it is almost impossible to get conformance between the process model and the real process.

2.2.3 Phase-independent Software Engineering Activities and Techniques

Management of the software process is an on-going activity through the whole life-cycle of a software system. We will look further into some of the aspects of managing the development of software according to the lower part of figure 2.3. Note that software process technology is described in a separate section (section 2.3).

Software Configuration Management

Software Configuration management (CM) is the development and application of standards, and procedures for managing an evolving system product. CM is typically concerned with preserving consistency of documents and source codes in software projects. Also the CM tools track and store the changes, making it possible to "go back" to a stable state of the process. CM tools are concerned with versioning products, and checking out files from a database to a workspace and vice versa. Simple CM tools such as SCCS [Roc75] and RCS [Tic85] use simple locking mechanism to ensure consistency, if two developers try to change the same file. By using strict locks, developers might have to wait before accessing files delaying the project. More advanced CM-tools such as ClearCase [Leb94] allow multiple developers to change the same file. This is achieved by creating parallel version branches, that later can be merged by advanced merging tools. In addition to CM tools, procedures for how to handle product changes should be defined. In larger companies, change control boards are used to approve changes for products and documents. This means that the product management is controlled by process management.

Software Cost Estimation

Software costs can be tracked to three main sources: 1) Hardware and software costs (including maintenance), 2) travel and training costs, and 3) effort costs (the costs of paying software engineers). For most software projects the effort cost is dominant. According to [Som95], there are five techniques for estimating software costs:

1. **Algorithmic cost modelling** developed by using historical cost information that relates some software metric to the project cost. An example of an algorithmic cost model is the COCOMO model [Boe81] that has been derived by collecting data from a large number of software projects.

2. **Expert judgement** where several experts on the proposed software development techniques and the application domain are consulted, and their estimates are compared and discussed.
3. **Estimation by analogy** where cost is estimated by analogy with similar completed projects.
4. **Parkinson's Law** where costs are determined by available resources rather than by objective assessments.
5. **Pricing to win** where costs are estimated to whatever the customer is willing to spend on the project.

Software Quality Management

The term software quality has different meaning to different people. According to [Fen91], a definition of the term software quality could include fitness for purpose, conformance to specification, degree of excellence, and timeliness. However if it should be possible to measure software quality, quality should be specified according to a selection of product attributes of interest to the user. What these attributes are is dependent on the application area of the products.

To goal of quality management within software companies is to produce products with a certain level of quality. Quality management typically involves these activities:

1. **Quality assurance** is the process of establishing a framework of organisational procedures and standards to enable production of high-quality software. Quality assurance standards have been defined for software products and for software processes by national and international organisations like US DoD, ANSI, NATO, and IEEE. Product quality standards are concerned with how the products are documented through consistent appearance, structure and quality. Some product standards also define documentation process including quality checks. The process quality standards define the process that should be followed during software development, and are most commonly used in the development of military software and for development of software for space shuttles and air-planes. One example of a quality standard for software is ISO-9001.
2. **Quality planning** is the process of selecting the appropriate procedures and standards and adapt them to a specific software project.
3. **Quality control** is the process of controlling that the quality procedures are followed by the software development team.

Software Process Improvement (SPI)

In the 1980s, the software industry became more and more concerned with quality. Inspired by Japanese factories that had well-defined processes to achieve better quality, international standardisation organisations also developed standards for the software industry.

ISO-9001 is a well-known international standard requiring documentation of a set of procedures for a software company to produce quality products. In order to be ISO-certified, a software company can systematically document the company procedures in 19 pre-specified work areas. This means that ISO-9001 can help a company to improve the software process to a certain level, but it does not specify a continuous improvement. A criticism of ISO certifications is that companies only use them to show customers their ISO-9001 diploma. Often the daily practices of a company are totally different from what their quality system prescribes.

The Capability Maturity Model (CMM) focuses on how software companies can improve their software development process. In CMM, the capability maturity level of the software development processes for a company is assessed. CMM defines five maturity levels, and recommended practices are described for each maturity level. Companies in the first maturity level do not have a defined development process at all. At the highest maturity level, the development process is continuously improved. The aim of CMM is to achieve a higher maturity level by improving the process. A major problem with CMM is that it is not very suitable to smaller software companies. Here some of the recommended practices can not be used because they are too expensive or hardly relevant at all. It has also been observed that in some cases that CMM instead of improving the efficiency of the organisation, has resulted in increased bureaucracy.

2.3 Software Process Technology (SPT)

Software process technology aims at giving process participants various kinds of support throughout the software engineering process. Software process tools are used to provide process guidance, for enabling process automation, for making process analyses, and for understanding the process better in order to improve it. When such tools are combined in an environment to help developers to produce better software with less time and effort, we call this a *Process-centred Software Engineering Environment* (PSEE). A PSEE (see section 2.3.1 is regarded as a software development environment that can be tailored to support specific software projects. PSEEs are different from other software engineering environment because the process is modelled explicitly, while e.g. in many CASE tools the process is implicit. The main component of a PSEE is a *process engine*, PSEEs may also consist of process modelling tools, process model compilers, process model analysers, product and model repositories, product tools etc. The process engine is responsible for executing the process model including interacting with the user, activate tools, and interchange data with a repository. There are two main types of software process models:

- **Generic models** are models of general software engineering processes that e.g. are available in text books and in quality assurance specifications. These models are often high-level and cannot be directly used to as a guideline for carrying out projects.

- **Specific models** are models that reflects one project.

The generic models are often used as a starting-point, but must be adjusted and detailed to create specific models and plans that can be applied directly in projects. To use specific models in a PSEE the models must be represented in some formal language, making it possible for the process engine to interpret and execute the model. PSEEs are used for different purposes, and these can be classified as following [BNF96]:

- *Passive role*. The user guides the process and the PSEE operates in response to user requests.
- *Active guidance*. The PSEE guides the user through the process by reminding the user what to do and when. The user is not forced to perform the actions.
- *Enforcement*. The PSEE forces the user to act according to a specified process model.
- *Automation*. The PSEE executes activities without user intervention.

The list above describes different views on how to manage software processes. Automation is a way of speeding up the software process, but cannot be applied to the most time consuming tasks of software development such as designing and coding software, and writing specifications. Enforcement is a way for the management to get the software developers to follow a defined process model that has been planned in advance and in detail (better conformance). In practise it is difficult to use PSEEs that enforce the process, because most software processes change during projects and generally most people do not like to be forced. Active guidance is a more relaxed process support, where the focus is on helping the software developers through the various steps of the process by providing useful information, and reminding the user when to do specific tasks. When a PSEE has a passive role, it will act as a process information system. Note that different kinds of enactment support can be provided in the same PSEE. In general, we can use PSEE to automate simple stable tasks such as configuration management, compiling and building programs etc, whereas guidance can be used for more creative and evolving parts of the process.

Process support in the CAGIS PCE

In the CAGIS PCE, we have chosen to use active guidance to provide process support. The CAGIS SimpleProcess workflow tool presents an agenda of activities and deadlines for the user, and it gives the user an opportunity for navigating through the activity network. Parts of the software processes can also be enforced in the CAGIS PCE, e.g. when software agents forces one party in a resource negotiation process to deallocate some resources. Enforcement of the process is however dependent on how the cooperation protocols are implemented.

Process Modelling Language (PML)

A Process modelling language (PML) describes processes through activities, products, roles, tools, directions, and performers as described in section 2.1.1. Further, PMLs can be classified in four main paradigms:

1. *Programming language based PML*, where the PML is a specialised programming language based on conventional programming languages.

2. *Rule based PML*, where production rules are used to describe the software process. Activities are described by rules with a pre-condition, an action, and a post-condition. Rules have an associated role being responsible for the activity, and resources needed to execute the activity.

3. *Extended flow or network based PML*, where Petri nets or Statecharts have been used to model software processes.

4. *Multi-paradigm based PML*, where two or more of the above paradigms are combined.

A programming language based PML often requires more time spent on creating (implementing) the process models because the process has to be programmed in detail. An advantage is, however, that such PMLs are very flexible and do not put many restrictions on what can be modelled. The extended flow or network based PMLs and rule based PMLs often demand less time on modelling a process, since such PMLs are rather easy to represent partly or fully in graphical notations. The major disadvantage with such PMLs is that the modelling language puts limitations on what and how the process can be modelled. A multi-paradigm based PML can for instance use a programming language based PML to model parts of a process that need a high degree of flexibility, while simpler parts of the process can be modelled by a rule based PML. Note that the choice of PML also often reflects the purpose of the PSEE such as active guidance, automation etc.

PMLs used in the CAGIS PCE

The process models in the CAGIS PCE are expressed in three different PMLs (the motivation for this is described in section 4.3 in chapter 4. Individual processes are modelled in the CAGIS SimpleProcess in a <u>network based PML</u> (number three in the list above) expressed in XML syntax. In this PML, the process is represented as activities with pre-order relations that are activated when a user push a button to notify that she/he has completed an activity. Cooperative activities are in the CAGIS DIAS modelled through an agent Java API. Although the agents are implemented in Java, we could call this a <u>programming language based PML</u> (number one in the list above). The CAGIS agent Java API provides a number of high-level methods that can be used to do agent specific tasks. However most programming language based PMLs deviate more from the original programming language. The third PML in CAGIS PCE, is CAGIS Glue modelling language defining relationships between workflow elements (process fragments) and software agents. The GlueModel can be regarded as being a part of the process model because it triggers events in the process. The GlueModel can be classified as a rule based PML (number two in the list above), because it specifies a set of rules that can trigger some actions. If we look at the CAGIS PCE as a whole, it uses a multi-paradigm based PML (number four in the list above). We have used three different PMLs, we can keep each PML relatively simple and the combination of them can cover different domains like simple individual process guidance (CAGIS SimpleProcess), cooperative process support (CAGIS DIAS) and dynamic process rules (CAGIS GlueModel).

2.3.1 Process-centred Software Engineering Environments (PSEEs)

In the late 1980s and early 1990s, many prototype PSEEs were developed. Most of these prototypes were non-commercial, and available for free for experimentation. Some of these prototypes where Adele [BEM91], ALF [B$^+$89], Arcadia [TBC$^+$88], EPOS [HC$^+$88], Marvel [Kai90a], Merlin [PSW92], OIKOS [ACM88], SPADE [BBFL94]. There were also some commercial PSEEs: IPSE 2.5 [War89], Syner Vision [HP93], and Process Weaver [Fer93]. All the PSEEs above had all their own PMLs and can be classified according to table 2.2. In Adele the process models were specified using a simple imperative language tailored for database access combined with rules (triggers), ALF combined logic programming with a rule based PML, and EPOS used an object-oriented programming combined with rule-based models.

Programming based	**Rule based**	**Network based**	**Multi-paradigm**
Arcadia	Marvel	Process Weaver	Adele
IPSE 2.5	Merlin	SPADE	ALF
Syner Vision	OIKOS		EPOS

Table 2.2: PSEEs classified according to PML paradigms

Within the **Software Process** community, research on how to support distributed, heterogeneous and cooperative software processes has recently gained more attention. In [Tia98], Tiako outlines how to model federation of PSEEs. In such a federation, large development projects can be realized by dividing the whole process into 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 [BSK95], 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. This approach is called a master-slave federation. In [BCN$^+$96], Basile et al. take Oz as a starting point to provide federated PSEEs, and allow 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.).

APEL [DEA98] developed at Laboratorie Logiciels, Systemes Reseaux, France is another research effort focusing on interoperability among heterogeneous PSEEs. In APEL, several process engines can co-exist in the same environment using a common state server for all process engines. This architecture makes it also possible to provide several different user interfaces. APEL aims to pursue two main goals: Support manage complex distributed processes in a heterogeneous environment, and support process evolution.

Senitel [CNF98] is a PSEE that allows people explicitly to deviate from a process model to cope with unexpected events. Traditionally this problem has been supported through changing the process model on-the-fly in order to deal with the unexpected events. Senitel specifies the process in the LATIN PML as a collection of *task types*. Each task type describes an activity as a state machine and is characterised by a set of state variables, a set of transitions, and a state invariant. State variables determine the structure of the internal state of a task type. The state invariant is a logical predicate that has to hold during the process. LATIN specifies two types of transitions: *normal transitions* and *exported transitions*. A normal transition is automatically executed as soon as its preconditions are true. An exported transition is executed if the user requests it and its preconditions are true. It is however possible to force an exported transition regardless of the evaluation of precondions. When the transition *fires illegally*, the user can deviate from the process model to deal with unexpected events. Senitel records the relevant events occurring during enactment in a knowledge base. The enactment is suspended only if one of the invariants is violated. This event will initiate the reconciling activity that used the knowledge base to perform a pollution analysis. A pollution analysis identifies illegally fired transitions and potentially polluted variables through a logical reasoning on the knowledge base.

The ESPRIT **PIE** project (Process Instance Evolution, project no. 34840 in 1998–2000 [CDE$^+$00]) attempts to federate several PSEEs by sophisticated and partly reflexive messaging facilities (middleware). PIE distinguishes between federations of type master-slave (either pre-planned or not) and peer-to-peer (no master, so pre-planned). In [BCN$^+$96], mechanisms and policies for federated PSEEs are discussed. The motivation for this work is that software is developed across several organisations. Among these organisations interaction can be characterised by:

- *Geographical distribution* where the organisation can be centralised or distributed,
- *Homogeneous vs. heterogeneous processes* where processes can be the same or differ from team to team,

- *Homogeneous vs. heterogeneous technologies* where the same or different technologies can be used for different teams,
- *Single company vs. multiple companies*, and
- *Singe nation vs. multiple nations* where companies have to overcome problems with different languages and cultures.

In a federation, levels of abstractions are used to differ between areas of concern. Basile et. al., have defined four abstraction levels in a federation:

1. *Infrastructure* typically TCP/IP, DCE, and CORBA,
2. *Basic operations* built on top of the PML/PSEE as specific process fragments,
3. *Inter-organisation policies* directly implemented as constructs provided by PML, and
4. *Inter-organisation process* providing support for definition and enactment of inter-organisation processes.

CAGIS PCE related to PSEEs

During the work with the CAGIS PCE, we have been looking at some of the same problems that have been addressed in projects link Oz, APEL, and PIE. These two projects have investigated how to federate several PSEEs. In Oz, this problem has been solved from a transaction (database) point of view through the *commit* activity where shared data are coordinated. In the APEL and the PIE projects, the problem of providing a federation architecture of heterogeneous PSEE has been addressed by looking at how can different process engines co-exist in the same environment, and how to deal with process evolution in a federated environment. One goal for the CAGIS PCE has been to provide a middleware for defining relationships and initiate interactions between different process support components. This middleware can also be used to federate other systems and to initiate cooperative activities between these systems. Since federation of PSEEs has not been the main focus of this thesis, we have not time to test and implement scenarios where federated PSEEs participate.

Senitel allows people explicitly to deviate from a process model supported through a framework with a formal basis. In the CAGIS PCE we allow people to change their processes during the enactment to provide flexibility, but we do not provide a environment for doing controlled process deviation (in CAGIS PCE this is ad-hoc).

2.3.2 Research Trends and Challenges

Software process technology has not been widely accepted by the software industry yet, and there are very few commercial PSEEs available. Does this mean that there is no need for SPT? If a software company wants to reach maturity level 3-5 in CMM (see section 2.2.3), they must focus on how they develop and maintain software (the software process). The software process must be defined (level 3), it must be managed (level 4), and finally it must be optimised (level 5). SPT provides tools for *defining* processes (modelling processes), for *managing* processing through process measurement and analysis, and for *controlling* the process through monitoring and automating the process. One reason for the little application of SPT in the software industry can be that SPT tools and models are not standardised. One very important research challenge is to find common representations for modelling processes, common software process architectures, and common software process tool APIs.

In order to use SPT, the software process has to be modelled in a formal model. Traditionally the process models have been created based on experiences from e.g. project managers, but more detailed information is needed to give process participants useful process support. A research challenge is therefore to *understand* what people are doing when they are developing software. Some recent research within SPT community has started to create tools for *logging* how software developers work [CW98]. These *process discovery tools* must be integrated with COTS tools like Microsoft Office, Visual Basic, Rational Rose,etc. , to really discover the details in the software process.

Another direction of recent SPT research has been (as mentioned in the previous section) to enable PSEEs to be run in heterogeneous environments, and to federate different PSEEs and SPT tools. To solve problems with heterogeneity and federation, standardisations of PMLs, process architectures, and process APIs are needed. Another approach is to create middleware that can bridge between existing tools and models.

According to Robert Balzer's keynote speech at the Seventh European Workshop on Software Process Technology [Bal00], the two main problems with existing SPT are that it does not give appropriate *cooperative process support* for software developers and it does not cope with *process changes* occurring during the process execution. This view is further supported by Cugola and Ghezzi in [CG98] and by Conradi, Fugetta and Jaccheri in [CFJ98], saying that process programming has failed to accommodate the parts of software development that involve humans. The emphasis of most PMLs and PSEEs has been on describing process models as *normative* models, containing pre-defined and expected activities, and pushing automation to enforce them. Although the intention has been to establish good practices for developing software, the official development process (the process model) is quite different from the actual process (poor conformance). That is, software processes are human-centred processes where interaction and cooperation are very central aspects of the process. This has often been ignored by many PSEEs. Cooperation is needed when negotiating about requirements or about resource allocation, when specifying and implementing the products, as well as during product testing. Human processes demand flexibility, in order to cope with creative tasks, as well as changing requirements, technologies, or work environment such as staff turn-over. A PSEE should therefore be able to handle situations where projects must be heavily re-organised or reformed during project execution.

The main focus in this thesis has been to find the required architecture, the tools and the models to better model and support cooperative and creative processes in a flexible way. These problems have not been addressed adequately in existing PSEEs.

2.4 Workflow

The term workflow is a term that is associated with factory automation at the time the second industrial revolution took place (around 1900). Around the 1950s, the same techniques were used to automate offices. The first Office Information Systems were developed in the 1970s, with mixed success. Most systems failed because they were too rigid and disturbed the working processes, rather than supporting them. Office Information Systems have been focusing on making office work efficient. Workflow has also been influenced from CSCW research. In the CSCW research field, a workflow system has been seen as a specific type of groupware to support collaboration between people. Here the focus has been on flexible computer support, as opposed to forcing people to work in a specific way.

Workflow Interoperability

The Workflow Management Coalition has worked out a specification for interoperability workflow binding in XML [WfM00], relying on an earlier, high-level process formalism (Process Interchange Format, PIF) and standard tool architecture. The intention of this XML specification is to allow workflow systems, supporting simple chained and nested workflows, to interoperate both synchronously and synchronously. In practice, this means that two workflow systems can exchange data and operations independently of implementation platform of these workflow systems. The XML specification describes how information in a *workflow message*, exchanged between workflow systems, should be written. Two types of messages can be used: *request* and *response*. A request message is used to initiate an operation in a remote resource, and/or to provide input to that resource. A response message is used to send the result of an operation to its requesting resource, providing output. The XML specification further defines how to specify process context and data, process status, error handling, and operation. Four operations are defined (either of type request and response):

1. *CreateProcessInstance* is used to instantiate a known process definition,

2. *GetProcessInstanceData* is used to retrieve the values of properties defined for the given process instance resource,

3. *ChangeProcessInstanceState* is used to modify the process instance state, and

4. *ProcessInstanceStateChanged* is used to notify that a state change event has occurred.

The workflow interoperability standard above illustrates one example of an important difference between SPT and workflow. The workflow community has managed to implement standards and specifications making it possible for tools to interact regardless of whether the tools are research prototypes or commercial. Although there is on-going research for federating PSEEs, the SPT community has not defined any detailed implementation or model standards. Maybe this is the reason why there are so few commercial PSEEs.

2.4.1 Workflow Systems

The workflow community has the last decades been focusing on building systems that support distributed processes. In [YL99], 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.

Endeavors [HL98] is an open, distributed, extensible process execution environment developed at the University of California Irvine, and has been licensed by Endeavors Technology Incorporated. It is designed to improve coordination and managerial control of development teams by allowing flexible definition, modelling, and execution of typical workflow applications. XML is used extensively for implementation. There are five main characteristics for Endeavors:

1. **Distribution** – Support for transparently distributed people, artifacts, process objects, and execution behaviour (handlers) using web protocols.

2. **Integration** – Allows bi-directional communication between its internal objects and external tools, objects, and services through its open interfaces across all levels of the architecture. Multiple programming languages are also supported through APIs.

3. **Incremental Adoption** – Components of the system (user interfaces, interpreters, and editing tools), may be down-loaded as needed, and no explicit system installation is required to view and execute a workflow-style process.

4. **Customisation and Reuse** – Implemented as a layered virtual machines architecture, and it allows object-oriented extensions of the architecture, interfaces, and data formats at each layer.

5. **Dynamic Change** – Allows dynamic changing of object fields and methods, the ability to dynamically change the object behaviours at runtime, and late-binding of

resources needed to execute a workflow process. Process interpreters are dynamically created as needed.

ProcessWeb [Yeo96] is a simple web-interface to ProcessWise [PMC96] Integrator produced by ICL in cooperation with the Information Process Group at the University of Manchester. ProcessWise Integrator creates an environment enabling the activities of people in an enterprise to be co-ordinated together, and integrated with the organisation's computing facilities. A process management system built using the ProcessWise Integrator has a client/server structure and consists of four main components: User Interface, Process Control Manager, Process description in their PML, and an Application Interface. The most important component of ProcessWise Integrator is the Process Control Manager (process engine), which acts as the central server. Its main function is to interpret the PML description of the process. To ensure that processes may continue indefinitely, the Process Control Manager has been implemented using a persistent store technology. ProcessWeb can also be regarded as SPT, because it has been used to model and enact software processes, and it has functionality for dealing with process evolution which is typical of software development processes.

ActionWorkflow [MG94] is a commercial workflow product from Action Technologies. ActionWorkflow inspired by speech-act sees a business process as a conversation between two particular roles; the customer and the provider. A business process is seen as a closed-loop ending with explicitly stated customer satisfaction. The basic structure for a workflow model in ActionWorkflow consists of four phases as shown in figure 2.5:

1. *Opening:* Initiated either by a *request* from the customer or an *offer* from the provider.

2. *Agreement:* The customer and provider agree upon unique *conditions of satisfaction* for the particular dialog instance.

3. *Performance:* The actual performance takes place here by completing the requested work and reporting of the completion.

4. *Acceptance:* The customer assesses the deliverable and declares satisfaction, or refuses to accept according to the agreed conditions of satisfaction.

The ActionWorkflow System consists of three main components:

- *ActionWorkflow Analyst;* This is the process modelling tool used to model business processes.

- *ActionWorkflow Application Builder:* This tool takes a process model as an input, and adds forms, fields, and interfaces to external systems.

- *ActionWorkflow Manager:* This tool is responsible for executing the workflow.

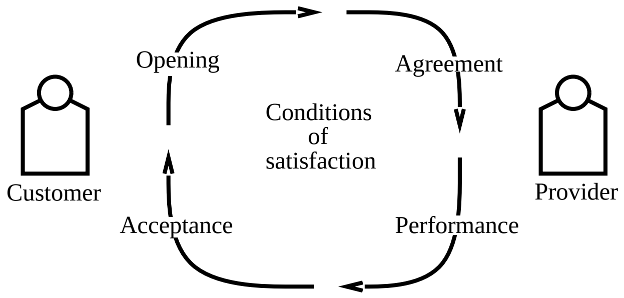


Figure 2.5: The four phases of a dialog in ActionWorkflow

In addition an ActionWorkflow API is provided to deploy client workflow enabled applications.

InConcert [Sar96] produced by InConcert Inc. is an open workflow tool based on object-oriented technology. InConcert provides distributed process support for various platforms. “To-do” lists are provided for the users, and the managers have process tracking and reporting tools available. The workflow can be changed during enactment, and ad-hoc routing of the workflow is also supported. A process is in InConcert represented as jobs containing jobs. A *job* describes a multi-person collaborative activity with some goal. The job consists of a task structure and a shared workspace. The task structure can be hierarchically decomposed, and a task has inputs and produces some outputs. Users are assigned to roles that are responsible for tasks.

CAGIS PCE related to workflow

The workflow systems presented above represents a selection of existing systems having some interesting assets. **X-MAS** is interesting because it illustrates an approach for combining workflow with mobile agents. In the CAGIS PCE, we use mobile agents to provide support for cooperative activities between people, while a more traditional workflow system is used to support individual workflow. In X-MAS, mobile agents are used to wrap activities that are distributed to move between people, tools, databases etc. **Endeavors** is another interesting workflow system because of its flexibility and its ability to run on heterogeneous platforms. Endeavors allows people to change their workflow during enactment, and does not put any restrictions on how the process is changed. We use the same principle in the CAGIS PCE, where people can change the process models during enactment in the way they want.

The implementation of the user-interface in CAGIS SimpleProcess workflow tool was based on experiences from the **ProcessWeb** workflow tool. The user-client in ProcessWeb is very flexible and convenient, since only a standard Web-browser is required to run it. However, compared to the CAGIS SimpleProcess tool, ProcessWeb has a very flexible programming language based PML that is very powerful for modelling interactions between people. **ActionWorkflow** and **InConcert** are examples of commercial workflow tools with two different philosophies. ActionWorkflow uses a PML where all activities are modelled as a conversation between the customer and the provider. The conversation based PML enables ActionWorkflow to model both activity networks and interactions between roles. InConcert uses an activity-network based PML, and offers a rich object-oriented framework for building specialised workflow tools.

2.4.2 Research Trends and Challenges

Recent research within the workflow field has been focusing on technical aspects as well as more fundamental issues. Typical technical challenges for workflow research have been to provide workflow systems that are scalable, have high availability, are easy to manage, and that can manage security sufficiently. Further, research within the workflow field has investigated how to provide distributed workflow, how to provide web-based workflow, how workflow can be embedded into existing software systems, how to use advanced transaction models in workflow systems, and how workflow systems can utilise technologies like CORBA and DCOM. Many of these technical problems have been solved in research prototypes, but in many commercial workflow tools these issues have not yet been addressed.

For the more fundamental research challenges, much research is still required to solve the existing problems. Workflow systems have suffered from the same problems as PSEEs by being too rigid when enforcing people to work according to a workflow model. Some recent workflow research has started to look at how flexible workflow can be achieved by allowing users deviate from the workflow model through exceptions, and by allowing process changes during enactment like Endeavors described in 2.4.1. One problem with flexible workflow is how to keep the workflow consistent. Deviation and process changes should be analysed to understand the consequences of the changes. Another research challenge has been how to offer users different views of the process from the same underlying workflow model.

CAGIS PCE and Research Challenges

The work with the CAGIS SimpleProcess workflow tool has addressed both some technical issues like how to provide distributed and web-based workflow, and more fundamental issues like how to support and model flexible workflow [Wan99, Wan00b]. The main difference between our prototype and other workflow prototypes is that we provide support for *moving a process fragment* (one or more activities in a workspace) between workspaces. This movement of activities can be used for re-allocating activities for different roles/people, for re-structuring processes, for re-arranging activities, and for building the processes from existing process fragments. This approach can be called *component-based workflow*, since the process fragments become individual components that can be combined in a flexible way. It is important to note that the CAGIS SimpleProcess workflow tool should be used in combination with our CAGIS DIAS agent system (support for cooperative activities) and the CAGIS GlueServer. The GlueServer will then execute a

GlueModel specifying rules for when the workflow process should be changed, e.g. when a allocation agent detects that the required human resources for a specific task are not available.

2.5 Computer-Supported Cooperative Work (CSCW)

In 1984, Paul Cashman and Irene Grief organised a workshop of people from various disciplines, sharing an interest in how people work and how technology can support people's activities. The workshop was called "computer-supported cooperative work". One main motivation for doing research within CSCW was that the challenges in office automation were not primarily technical anymore. To improve cooperative support for people working with computers, we had to learn more about how people work in groups and organisation and how technology affects that. CSCW started as an multidisciplinary effort by technologists to learn from economists, social psychologists, anthropologists, organisation theorists, educators, and anyone else who can shed light on group activity [Gru94]. CSCW applications typically include desktop conferencing and video conferencing systems, collaborative authorship applications, electronic mail extensions, electronic meeting rooms, and group support systems. Workflow tools and process tools can be seen as a specific group support system.

2.5.1 Process Support for Cooperative Work

This section describes research that has been focusing on process support for cooperative work from a CSCW point of view. Note that this section gives examples of PSEEs and workflow systems that can be regarded as systems for CSCW.

Cooperative work involves three aspects:

- **Communication** between involved parties: often low-level control flow.
- **Coordination** where participants activities are planned, scheduled, and monitored, and where negotiation and arbitration also are necessary: i.e., high-level control flow.
- **Collaboration** where participants share and exchange data and plans: mostly data flow.

Cooperative systems and related support are also categorised according to the **time/location matrix** [Gru94] as shown in figure 2.6. The CSCW typology shown in the figure distinguishes between synchronous and asynchronous work, and non-distributed and distributed work. In addition, this typology differentiates between predictable and unpredictable synchronisation and distribution. Examples of types of CSCW systems are also shown in the figure.

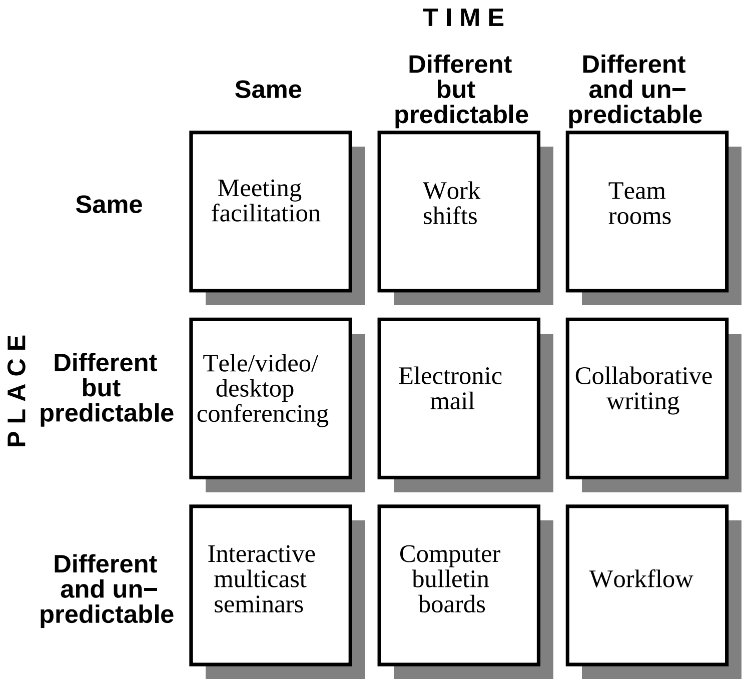

TIME
Same
Different but predictable
Different and un–predictable
PLACE
Same
Different but predictable
Different and un–predictable
Meeting facilitation
Work shifts
Team rooms
Tele/video/ desktop conferencing
Electronic mail
Collaborative writing
Interactive multicast seminars
Computer bulletin boards
Workflow

Figure 2.6: Cooperative systems categorised according to the time/location matrix

Most of the work in the software process and workflow community has been focusing on how to make people work together in an organised and model-based way (partly pre-planned). For high-level processes with little detail, it is possible to make people work in this manner. However, the development of software – or creative work in general – involves that people cooperate closely and negotiate to solve problems and to do the actual work. These kinds of processes are very hard to support by traditional software process support tools, because the focus will be more at cooperative aspects than pure coordination of work [WLCM98a].

Much CSCW work has been characterised as being synchronous and distributed according to the above time/location matrix. We may add an extra dimension to the CSCW typologies, considering different kinds of cooperative work, put in increasing complexity of the process support they need [LC98]:

1. **Ad-hoc cooperative work** such as brainstorming, cooperative learning, informal meetings, design work etc. Process support is often implemented through interactive blackboards and awareness triggers, such as in [DT93] and Ariadne [SD97]. For instance, there is little formal product modelling to explain why certain people are “related” and therefore need to cooperate.

2. **Predefined/strict workflow**, in traditional Office Automation style is often repre-

sented by simple document/process flow. Examples of such systems can be Lotus Notes [Orl92], Active Mail [GSSS92] and MAFIA [LvRH90].

3. **Coordinated workflow**, such as traditional centralised software maintenance work consisting of check-out, data processing, check-in, and merge steps. There exist several systems supporting coordinated workflow (mostly prototypes), e.g., EPOS [CHLN94], MARVEL/Oz [BSK95], and APEL [DEA98].

4. **Cooperative workflow**, such as decentralised software development and maintenance work conducted in distributed organisations or across organisations. Here the shared workspace and the cooperation planning are the main extra factors from the process point of view. Example of a system supporting distributed organisations and processes is Oz [BSK95], and Endeavors [HL98].

According to the classification above, the CAGIS PCE should be placed into the category *cooperative workflow* since this environment provides support for cooperative, distributed software development. In addition, is it possible to categorise one of the components in the CAGIS PCE differently. The CAGIS DIAS (agent system) is suitable for providing process support to *ad-hoc cooperative work* such as brain storming, meeting support, and awareness.

Lastly, it must be said, that CSCW has had a tradition for involving social scientists – such as psychologists, social anthropologists, and organisational theorists – in carrying out realistic (in-vivo) studies.

2.5.2 Research Trends and Challenges

Recent CSCW research has been focusing on various aspects of cooperative software systems, such as how to provide virtual communities for people that work in a distributed environment. Systems providing support for virtual communities allow people to work in virtual office environments where people share information, talks, and interact in other ways. Schmidt and Bannon in [SB92] argues that the lack of focus in CSCW may hinder its further development and lead to its dissipation. They further suggest that CSCW should be concerned with the support requirements of cooperative work arrangements. *Awareness* has been another important research area within CSCW, where the focus has been on how to provide efficient infrastructures and user-interfaces making people aware of other people and events. The *World Wide Web* and the Internet have introduced new possibilities and research challenges for the CSCW community. Many collaborative systems to be used on the Web have been made both as research prototypes and as commercial systems. The Web and the Internet have opened a wider possibility to share information and to interact beyond geographical distances and time-differences. Recent CSCW research has also focused on pure *technical issues* like how to use Java and component architecture to develop systems for CSCW. In addition the focus has been on building open systems that can interact with a variety of tools, and that can run on different platforms.

Maybe the biggest challenge for research concerning process support for cooperative work is to find the right balance between *forced behaviour* (automation), and *non-unobtrusive support* (e.g. notifiers). Traditionally, the CSCW community has opposed all forced behaviour, while the SPT and workflow community has been focusing too much on automation. Also groupware created from CSCW research has not been focusing on process support at all. These tools have provided an infrastructure for doing cooperative work (for e.g. cooperative editors, sharing of documents etc.), without giving any process support at all. In order to find the right balance, the appropriate types of cooperative processes must be identified and characterised. For each type of process, various types of process support should be identified.

CAGIS PCE related to CSCW

The cooperative support in CAGIS is based on **agent meeting places** (AMPs), called **agoras** in [MDP98], where standard or specialised agents can meet to help solve or negotiate about a given task. Note, that such meeting-places are different from conventional, controlled workspaces with formally checked-out artifacts. The process support in CAGIS (including the GlueServer, CAGIS SimpleProcess workflow tool, and the CAGIS DIAS agent system) can also be viewed as a framework for cooperative workflow, since the focus is on support for distributed processes with cooperative aspects. Our approach combines CSCW-like cooperative agents with traditional workflow support. Our research distinguishes itself from other CSCW research with the focus on process.

2.6 Software Agents

The history of software agents started in artificial intelligence research in the late 70s, where Carl Hewitt worked with a concurrent *actor model* [Hew77]. He proposed self-contained, concurrent, interactive executing objects called *actors*. An actor had a encapsulated internal state and could respond to messages from other similar actors.

It is said that software agents are a new paradigm for developing software applications. This has made research fields like software architecture, middleware, artificial intelligence make use of software agents as a part of the research. Software agents can be applied in a wide range of domains, like web-search agents, email-filters, desktop helpers, advance control systems, e-commerce, cooperative computer support etc. In this section, we will focus on two types of agents that are relevant to the CAGIS multi-agent architecture described in this thesis; namely mobile agents and cooperative agents. These two types of agents are combined in the our work.

2.6.1 Mobile Agent Systems

A mobile agent is an agent with the ability to move around in a network. Mobile agents have become a very popular research topic lately, and is also a much mis- or overused word. Many technologies facilitate moving objects between hosts.

The **Aglets** framework [LO98, Lan97, OKO98, LA97], developed at IBM's Research Laboratory in Japan, but is not being supported anymore. Aglets provide support for running Java programs (code and state) to move around in a network from one host to another. That is, an aglet executing on a host can suddenly halt execution, dispatch (migrate) to another remote host and start executing again. There are often misunderstandings between the terms aglets and applets. Unlike applets, an aglet also carries its state when it migrates. An applet is only passive code that can move across the network from a server to a client. At a (new) client, the applet is initialised in the context of a run-time process where an interpreter can execute its code. An aglet can, because it is carrying its state wherever it goes, travel in sequence to many destinations on a network.

Voyager [Gla99] by ObjectSpace is a product family consisting of an Object Request Broker (ORB) with services, and an application server where the ORB supports mobile agents.

Odyssey [Whi96] developed by General Magic enables developers to create and debug their own mobile agent applications implemented in Java, and offers support for arranging meetings on particular hosts, and publications of objects.

Jumping Beans [Ad 99] builds on the Java platform and provides a framework for Java programs to "jump" from computer to computer. The Jumping Beans architecture is based on a client-server architecture, where programs moving from one host to another must do this through a central management server. The central management server has a very strict security system. This implies that the Jumping Bean framework is best suited for small distributed networks rather than WANs like the Internet (because of firewalls).

Grasshopper [IKV00] is an agent development platform launched by IKV++ in 1998. It enables users to create agent applications enhancing electronic commerce applications, dynamic information retrieval, advanced telecommunication services and mobile computing. Grasshopper is completely implemented in Java and provide communication services through CORBA giving the benefit of high distributed integration.

Bonding and Encapsulation Enhancement Agent (Bee-gent) [Cor00c] is an agent development framework developed by Toshiba Corporation. It is entirely based on the mobile agent concept. All applications become agents, and all messages are carried by agents; thus it is possible to build a distributed system using existing applications

All the mobile agent frameworks above uses Java as an implementation language for agents. Voyager and Grasshopper are a bit different compared to the other agent frameworks, since they use an ORB that supports mobile agents, making it possible to integrate other systems through standard CORBA interfaces.

2.6.2 Cooperative Software Agent Systems

In multi-agent systems, the software agents must cooperate in order to reach their goals. In such systems, the agents are autonomous entities that can perform tasks assigned to them independently, without user intervention. Such agents can be used to negotiate about resources in domains like distributed support systems, system management, artificial intelligence, and electronic commerce.

The **DYNAMICS** agent framework described in [TGML98], Tu et al. present an architecture to integrate mobile and intelligent agents providing an approach of dynamically embedding negotiation capabilities into mobile agents. One of the main difficulties with a practical implementation of a system combining intelligence and mobility, is to prevent that the agents get too big. Intelligent agents often must hold much more code than e.g. simple GUI mobile agents. To deal with this problem, Tu et al. have listed three important requirements for the design of plug-in[2] architecture for mobile agents:

- *Role-specific functionality:* A mobile agent should only carry the functionality to fill the role it is assigned to do.
- *On-the-fly loading:* Functionality needed of an agent should be loaded ”on demand”.
- *Flexible configuration:* The agent's functionality should also be flexible and dynamically configured, in order to reuse functionality in similar, but different constrained situations.

Tu et al. further proposes that the design and implementation of this architecture results in a plug-in architecture called **DYNAMICS**, with the following components:

- *A Communication module*, concerned with the delivery and processing of any kind of messages exchanged between the agents. If a message is recognised as a negotiation message, its content is passed to the protocol module. KQML has been used for implementing the communication module, and a new ontology called *negotiation* has been introduced to recognise negotiation messages.
- *A Protocol module*, responsible for the protocol compliance of an agent. This implies that the content of each incoming and outgoing negotiation message is inspected by the protocol module. The interface between the communication module and the protocol module is determined by:
  In: incoming negotiation message content and sender of message, and
  Out: outgoing negotiation message content, addresse(s) of message and/or sending mode (unicast/multicast)

[2]Plug-in means components that can be added and removed dynamically

- *A Strategy module*, implementing a negotiation strategy responsible for producing proper negotiation actions as required by the protocol module. The following dynamically typed parameters are used:
  In: Current negotiation state is passed from protocol module.
  Out: Negotiation action computed by the strategy module.

- *A Rule module*, used to configure or constrain objects. Configuration can be achieved by performing actions on the main properties of the configured components. Three different rule types exist: *Invariant* - is a condition that must hold true at any time with regard to the entity it is assigned to (one or a group of cooperating agents), *Policy* - consists of a goal and an action leading to the goal, and *Action rule* - consists of a trigger and an action to be performed when the trigger holds.

This structure defines the functionality of each module through interfaces, making it possible to use different implementation methods or algorithms. In this DYNAMICS architecture, most of the application semantics of mobile agents is realised by plug-ins. The following types of plug-ins can dynamically be incorporated into agents:

- *Roles* introduce new functionality into agents or entities which serve as role containers. In a business environment, roles can typically be ”sellers”, ”buyers”, or ”notary”.

- *Substitutes* are used to provide new implementations of or replace an existing implementation of an interface.

- *Configurations* are used to reconfigure an application component dynamically, meaning that both the interface and implementation of the reconfigured component remain unchanged.

The interconnection between the different modules is as follows: The communication module is a role plug-in, the protocol module is a substitute plug-in called by the communication module, the strategy module is a substitute plug-in called by the protocol module, and the rule module is a configuration plug-in for the strategy module. DYNAMICS is used to create mobile, cooperative agents, and it is implemented in Java using Objectspace’s Voyager system.

The **DESIRE** agent framework described in [BvET97], shows how a multi-agent modelling framework can be used to model competitive cooperation of agents. DESIRE provides support for the design of a conceptual model of the behaviour of interacting agents. Compositional agent models define the structure of the architectures, and components in a compositional model are directly related to agents and their tasks. In DESIRE, five types of knowledge are represented at a conceptual level:

1. *Compositional structure* of agents and their tasks. Tasks can be decomposable or not, and are characterised by their input and output knowledge structures.

2. *Interaction* within and between agents and tasks.
3. *Temporal relations* between tasks (represented by rules in temporal logic).
4. *Delegation* of tasks to agents.
5. *Knowledge structures*.

Cooperation is an effective approach for allocating limited resources in many real life situations. Brazier et al. use housing as a limited resource as an example of a real life scenario where competitive cooperation can be applied. In the allocation process of apartments, people search for housing with a monthly rent between x and y. Real estate agents act as coordination agents that are responsible for conflict resolutions. When the DESIRE framework is applied to this scenario, the competitive cooperation is modelled with competitive agents and a material world where shared information is stored. A competitive agent contains the following tasks:

- *Maintain World Information.* This task stores the material world information, being information an agent has about the world state, namely presence of agents and resources.
- *Agent specific tasks.* One agent specific task is modelled, namely *Obtain Resource*.
- *Cooperation Management.* This task is responsible for four subtasks: 1) Update current agent information, 2) Determine access to resources depending on information about the world, priorities between agents, and co-operativeness of other agents, 3) Determine Priority based on domain specific knowledge, and 4) Determine Cooperation through observation and reasoning.
- *World Interaction Management.* This task is responsible for observing and performing actions proposed by *Obtain Resource* in the Agent Specific Tasks.
- *Agent Interaction Management.* This task manages communication between an agent and other agents.
- *Own Process Control.* These tasks are responsible for determining which information is needed to decide whether access to a resource is allowed, and where this information is found.
- *Agent task control.* This task specifies activation of the task Own Process Control.

The main difference between the cooperative software agent systems presented above is that DYNAMICS has combined mobility with intelligence, while DESIRE focuses on conceptual models of the behaviour of interacting agents. If it is required to have mobile agent support, DYNAMICS offers the right balance between mobile agents not being to fat (in code-size) and modules for intelligence that can be incorporated in the agent on demand. DESIRE's strength is that is has a conceptual model for solving tasks where agents have to compete to reach their goals.

2.6.3 Research Challenges and Trends

Considerable effort has recently been spent on research on software agents. However, still most of the existing agent frameworks are immature. One goal of software agent research must be to standardise software agent architectures. Within mobile agent research, effort has been spent on how to provide small efficient, intelligent mobile agents, and languages for specifying their coordination. In [CFM00], Ciancarini et. al. propose a coordination language that provides a formal framework for specifying and analysing mobile software agent. A coordination language can express the creation and destruction of agents, their communication activities, their distribution and mobility in space, and synchronisation and distribution of their action over time. For cooperative agents there is a need for standardising how they interact through standard procedures and formal models like speech-act and state machines. Another problem with software agents is that there exists almost no validation of such systems by end-users. This means that it is really hard to predict how useful agent systems really are. In [Han00], Hannemyr argues that implemented agents are much simpler than their counterparts in the literature. He indicates that it is really hard to implement very useful and advanced software agents, and that the "agent" name is often used to sell products instead of functionality. There are however some current developments of agent prototype systems that can deal with the problems of intelligence, adaptive reasoning and mobility. These systems may not meet the high and in some cases exaggerated expectations that agents can be used for solve anything, but it is a step in the right direction.

CAGIS PCE related to Software Agents

As a part of the CAGIS PCE, the CAGIS DIAS is a mobile multi-agent architecture that was originally implemented in Java by using the Aglets framework. By using mobile agents we could benefit from less demand on servers since agents can execute locally, and from less network load since the clients do not need to continuously interact with the server. The current implementation of DIAS provides an infrastructure for cooperation between agents through KQML, although the agent interaction does not rely on an underlying conceptual model for doing reasoning. A simple state machine in combination with speech-act protocols is used by the agent to decide what to do next. This ad-hoc approach has been sufficient for solving the simple negotiation processes for resource allocation present in the scenarios we have used. However, for supporting more complex cooperative situations (e.g. negotiations), a more formal framework should be used to model the agent interaction. The DYNAMICS framework presented in section 2.6.2 is an appropriate candidate here, because it combines mobile agents and intelligent agents. Another approach could be to drop mobility, and only focus on competitive intelligent agent cooperation. In this case, the DESIRE framework is the most desirable solution because of its focus on formal modelling of cooperating agents.

2.7 Middleware

Middleware products are typically used as bridges between software product, e.g. between a database system and a Web server. The term middleware is used to describe separate products that serve as the glue between two applications.

Common middleware categories include: TP monitors, DCE environments, RPC systems, Object Request Brokers (ORBs), Database access systems, and Message Passing. This section describes middleware technology that are relevant to this theses, namely Web-related technology, Object and component architectures, and Distributed Java access. For a general introduction to **middleware**, see [Ber96].

2.7.1 Middleware Systems and Technology

This section looks into examples of existing middleware systems and technology.

Web-related technology

The World-Wide-Web (WWW) has become the defacto standard for sharing documents and communicating in distributed environments. WWW has provided an easy, although simple way to make information or services widely available. Connected to the Web/Internet, universal web-clients will run a local web-browser that talks to a web-server, that again can talk to e.g. a database server over a CGI (Common Gateway Interface) protocol. There are now also many user-oriented, design tools for creating and maintaining web-pages and corresponding web-sites.

The Web relies on the **Hyper-Text Markup Language (HTML)** used to specify the look and the contents of web-documents. HTML is used to specify what colours, fonts, graphics, video and sounds that should be visible when looking at this web-page. HTML can also be used to specify simple interfaces through HTML-forms. These HTML-forms do not provide the full functionality of user interfaces as provided in graphical operation systems, but offer graphical user interfaces (GUIs) for doing selection from a list, entering text, selecting among radio-buttons, highlighting check-boxes and pushing buttons. More specialised user interfaces for e.g. showing tables are not supported. Also by using HTML, a web-page will be static. Because of these shortcomings, other web-standards are used in combination with HTML e.g., Dynamic HTML, Java script, Macromedia etc. HTML lacks the capability of specialisation. HTML describes *how* the data on a web-page is presented through markup elements, but does not “understand” *what* the data represents.

The **eXtended Markup Language (XML)**[Hol98, XML99] is more flexible, because we can define our own markup elements. This enables us to tailor a document (often a textual model in some formal language) to our own needs, storing and structuring the data (document symbols) as we like. We can add that many XML parsers and other XML support tools are available, and can be downloaded for free from the Internet. Initially, the WWW was meant to be a network, where documents could be shared for reading.

Recent the **Webdav** [WW98] initiative has provided a protocol and a framework for also access web-documents for editing. By using Webdav, it is possible to share documents on the web both for reading and editing. Webdav offers simple collaborative versioning of documents, but is has not gained a full intension from tool vendors. The last versions of Microsoft office have support for uploading and downloading documents from the Web by using the Webdav protocol.

Object and component architectures

In the hardware industry, the term component is used to describe a part of a device. A hardware product is typically an assembly of several components. Recently, the term component is also used for software parts that can be combined and assembled to produce a product. Components are reusable and have a well-defined interface. A possible definition of a component can be "*A small binary object or program that performs a specific function and is designed in such a way to easily operate with other components and applications*" [ISP00a]. This section describes component architecture providing infrastructure for distributed components to interact.

The Object Request Broker (ORB) concept and the **Common Object Request Broker Architecture (CORBA)** [OMG97, Bak97] are a central part of Object Management Architecture (OMA) defined by the Object Management Group (OMG). CORBA defines a mechanism that allows software components to communicate with each other and the system itself. The location of components living on a CORBA bus (software component bus) is totally transparent to the clients, and the clients can invoke a component's methods, both statically and dynamically. Static invocation means that the client knows the method name and its parameters. For dynamic invocation, the client will ask the server for information about available methods. All applications/objects communicate through the CORBA bus, and their interfaces are specified in the Interface Definition Language (IDL). The CORBA IDL is mappable to programming languages like Java, C, C++, Perl, COBOL etc. CORBA services are collections of system-level services packaged with IDL-specified interfaces, that complement the functionality of the ORB. OMG has published standards (some rather general) for several services, such as naming, events, life-cycle, persistence, transactions, concurrency control, relationships, externalisation, licensing, querying, properties, security, time, collections, and trading. The CORBA facilities and domain interfaces provide component frameworks that specify rules of engagement, common data formats, and architecture boundaries.

Microsoft Distributed Component Architecture (MDCA) [CRW98] is an architecture that is Microsoft's answer to the challenges that component software presents the computer industry today. Component Object Model (COM), Distributed COM (DCOM) [CRW96] and its ActiveX language, and the next generation COM (COM+) denote a set of related component object models incorporated into Microsoft's family of Windows operating systems. The "wrapped" components may also exist as binary code, not only as source code that is the case with CORBA. The term ActiveX do not refer to a well-defined technology, but can be characterised as a brand name. Today, Active X are most commonly used to denote ActiveX Controls, which are components that follow certain standards in how they interact with their clients. COM is not a technology for implementing components, but it is used to define components and their interfaces. In addition, COM defines how components and clients interact by providing means for clients to call methods in components through a well-defined interface. COM provides transparent access to components on a single host. DCOM extends COM and makes it possible for components to communicate across a network. COM+ makes it easier for developers to create and use software components in any programming language and by using any tool. COM+ also introduces several new services to components, such as publishing and subscribing, in-memory database with transaction support, queued components, and dynamic load balancing.

Distributed Java access

Java has become the default programming language for programming distributed systems for the Internet. The major advantages in using Java are object-orientation, platform independent and rich support for distributed computing. This section describes methods and framework for making distributed systems using Java.

Java Remote Method Invocation (Java RMI) [Mic99a] is an interprocess protocol for Java, allowing Java Objects living in different Java Virtual Machines to invoke transparently each other's methods. Since these Virtual Machines can be running on different computers anywhere on the network, RMI enables object-oriented distributed computing.

JINI technology [Edw99, Mic00] is designed to enable users to simply connect any number of digital devices to the network, creating a "plug-and-play" community. By using JINI technology, the network itself can be very dynamic; devices and services can be added and removed regularly. JINI provides mechanisms to enable smooth adding, removal, and finding devices and services on the network. JINI is built on top of Java, object serialisation and RMI, which enable objects to move around the network between virtual machines.

JavaSpaces [Mic99b, FHA99] from Sun is based on JINI, and is a framework for dynamic communication, coordination, and sharing of objects between network resources like clients and servers. In a distributed application, JavaSpaces technology acts as a virtual space between providers and requesters of network resources or objects. The virtual space serves as a shared, network-accessible repository for objects. This virtual space allows participants in a distributed environment to exchange tasks, requests and information in the form of objects located in different JavaSpaces. The communication is handled by matching objects in a Space with template objects; thus there is no need to know the address of a specified host. This means that JavaSpaces technology provides a novel programming model, that views an application as a collection of processes cooperating via the flow of objects into and out of one or more JavaSpaces. It differs from the traditional view where messages are passed between processes to invoke methods on remote objects.

2.7.2 Research Challenges and Trends

Middleware technologies can be used as a technological base to support distributed cooperative software engineering. It is, however, not possible to simply pick the necessary technologies one by one, combine them, and obtain a required distributed system.

To provide a graphical user interface (GUI) that is accessible for users in a heterogeneous environment, HTML can be used. Although, HTML has limited capabilities for providing advanced user interfaces, most user interaction can be supported through HTML. If more advanced GUI features are needed, Java applets can be embedded into a web-page. Proper methods to develop and maintain Web-based systems are however still immature. XML is suitable for representing data, independent of how the data are represented in a web-browser. The transition from XML to HTML must be efficient, as well at it should be possible to change the way the data are presented without redefining the XML. A research challenge for XML and HTML is to validate existing XML/HTML methods and tools. XML-tools available are, however, very useful for fast prototyping.

In [ABE00], Andersen et al. argue that traditional middleware like CORBA and Java RMI are not flexible and adaptable enough for new application types such as multimedia, real-time and mobility. Middleware must be configurable to satisfy requirements such as scheduling policies, special protocols for multimedia and resource management. Further, they suggest that reflective middleware should be used to allow dynamic configuration of middleware.

In a distributed execution environment it is important to enable efficient interaction between various software and hardware parts. Object and component frameworks try to provide such interaction through a software bus. By using a component framework, reusability can also be provided, but this is no automatic feature. Such component frameworks should also give guidelines for how to implement components that are reusable in future implementations.

CAGIS PCE related to Middleware

This thesis does not address any research challenges described above for middleware, but middleware is used as an enabling technology to provide the architectures and tools we need. Our CAGIS PCE has used middleware extensively like HTML for providing user interfaces, XML for storing data and for data exchange, and CGI to provide an interface between the workflow tool and a web browser. Our work with a multi-agent architecture has required us to use middleware like KQML, Java RMI, JavaSpaces, and JINI. Further, we have used CORBA to facilitate a communication bus where other system can access the agent system.

There is also a novel **GlueServer** [Wan99, Wan00b], a new middleware for connecting static (pre-planned) workflow with software agents to achieve more dynamic (on-the-fly) cooperation [Bjø00]. The GlueServer consists of a GlueEngine, a GlueModel, an Agent Interface and a Workflow Interface. A GlueModel, written in XML, specifies how simple workflow processes involve interactive agents for handling cooperative and dynamic aspects of the process. A workflow tool will typically notify the GlueServer about the workflow process state. The GlueServer will check this state against the GlueModel, and possibly invoke agents (e.g. negotiation agents). The result (e.g. from the negotiation agent) will be brought back to the GlueServer, and a reaction (also defined in the GlueModel) will be sent back to the workflow tool. Typical reactions can be to re-start a workflow sub-process, change a workflow sub-process, halt a workflow sub-process etc. The GlueServer is implemented in Java, and by using a XML parser from SUN Microsystems, and ORBIX Web's CORBA implementation. The GlueServer can also be used as middleware for providing federation of other agent systems and workflow systems.

CHAPTER 3

Research Focus and Method

This chapter describes the research focus and method of this thesis.

3.1 Research Focus

The research focus for this thesis has been on how to provide flexible process support for cooperative software processes in heterogeneous environments. The research focus can be divided into three main parts:

- **Models and Concepts:** This part of the work has been focusing on what models and concepts are necessary to model and support processes that contain both distributed, cooperative processes, and local, individual processes. We found that it was necessary to divide between cooperative and individual processes. It was therefore required to specify the interaction between cooperative processes and individual processes. Our research in models and concepts has resulted in a process modelling language for individual processes presented in the paper "*Support for Mobile Software Processes in CAGIS*", chapter 17. Further, the paper "*Integrating Software Process Fragments with Interacting Agents* in chapter 18 describes a language for specifying relations between process fragments and agents.
- **Architectures:** To be able to perform distributed process support in a heterogeneous environment, an architecture had to be established. This architecture had to take care of the distribution of people and groups of people, and provide an efficient infrastructure for doing interaction between distributed entities. This infrastructure needed a facility to distribute documents and models, and also a way of

migrating these documents and models between distributed entities. The papers "*A Multi-Agent Architecture for Cooperative Software Engineering*" (chapter 14), and "*Design Principles for a Mobile, Multi-Agent Architecture for Cooperative Software Engineering*" (chapter 15) present our multi-agent architecture, and how it was designed.

- **Technology:** Since this PhD work had very limited human and time resources, it was important to (re)use existing technology that could offer the needed services. Typical technology was middleware technology for providing a distributed infrastructure, uniform user-interface clients (Web-clients), repository services, interoperability interfaces to other tools and systems etc. When choosing technology it was important that it was open, that it could be run on different machines and operating systems, and that it was relatively easy to program and configure. Functionality for fast prototyping has been regarded as more important than performance when selecting technology. The paper "*Implementing a Multi-Agent Architecture for Cooperative Software Engineering*" presented in chapter 16 describes the technology choices we made for our multi-agent architecture.

3.2 Research Methods

This section presents research methods that can be applied in software engineering research, and how this work relates to this thesis.

3.2.1 Introduction

Software engineering is a multi-disciplinary research field stretching from technical research like development of various tools, through modelling and languages, to more non-technical research like managing people and changing processes and organisation etc. Software development cannot be performed like manufacturing, because it contains human-intensive and creative activities. The goal of research within software engineering is to make tools, methods, and models enabling software to be produced more effectively, with better quality, on time, and spending less resources. From the definitions of software engineering presented in section 2.1.1, we recall that software engineering focus on the development of large software applications and that many people are involved in the development. This means that it is very time-consuming and expensive to perform full-scale research experiments, and in most cases this is impossible. Research methods are however required to examine if new technology, methods, tools, processes and so on really improve goals like the software quality, and less development time. In [Bas92], Basili has identified three main research approaches that are commonly used for doing experiments in software engineering:

1. **The engineering method:** By using the engineering experimental method, engineers build and test a system according to a hypothesis. Based upon the result of the test, they improve the solution until it requires no further improvement. The engineering method is typically used to find better methods for structuring large systems, and software engineering is here viewed as a creative task not to be controlled by anything else than necessary restrictions on the resulting product.

2. **The empirical method:** A statistical method is proposed as a means to validate a given hypothesis. Unlike the analytical method, there may not be a formal model or theory describing the hypothesis. Data is collected to verify or falsify the hypothesis. The empirical method can be applied on new technology to determine if this new technology is better or worse than the existing for producing software effectively.

3. **The mathematical method:** The mathematical method is based on mathematical and formal methods for doing experiments. A formal theory is developed and results derived from that theory can be compared with empirical observations. The mathematical method is usually used to find better formal methods and languages, where software development is viewed as a mathematical transformation process.

The engineering method and the empirical method can be seen as variations of the scientific method [Bas93]. Note also that a mixture of the research methods can be used together.

The research approach being used in software engineering is determined from the focus of the software engineering research. If the focus is mostly on technical aspects, the engineering method will be the most appropriate selection. A problem with technical experiments can be to preserve objective view of the experiment. When comparing a research prototype with other existing solutions, it is usual to “bend the truth” to favour own prototypes. Since it is very hard to test a number of systems in real environments, many short-cuts will be taken to get the results such as testing only parts of the prototypes, selecting few elements from the real environment that participates in the experiment etc. These short-cuts have a tendency to favour own systems.

In other research where the focus is more on the social and human aspects of software development, we tend to use research methods being used for social sciences. A problem can be to bridge results and research between social and technical software engineering research, because there is fundamental differences in how they work.

In the thesis the emphasis has been on the use of the engineering method, but we have also conducted some limited empirical studies. In in section 3.3), we describe more in detail the research methods used in this thesis.

3.2.2 Metrics Definition and Data Collection

Data collection is an important aspect of experiments to validate research. Before starting to collect data, it is important to know what metrics to be measured. Goal Question Metric (GQM) method [BCR94b] is an approach used within software process improvement, but can also be used to define metrics to be measured in software engineering experiments. GQM method starts by defining the goal for an object, in this case in an experiment. The next step is to establish a set of questions used to characterise the achievement of a specific goal with respect to a selected quality issue. Finally, the final step is to associate a set of data with every question in order to answer it in a quantitative way. In the final evaluation of our CAGIS PCE approach described in chapter 20, the GQM method was used to define the metrics that were measured in the experiment. The GQM method enabled us to more clearly specify the objectives of the experiment and how to measure these objectives.

When the metrics are defined, the data will be collected. By collecting ”wrong” data or by not collecting enough data, an experiment can be worthless. Here are some aspects that according to Zelkowitz [ZW98] should be considered when collecting data:

- **Replication:** It is important that other researchers can replicate the results of an experiment by reproducing the experiment. Unpredictable variables can make replication of results impossible and should be avoided. In addition it can be hard to get a homogeneous sample of subjects for all runs of the experiment. This effect can be counteracted by randomising the factors out of concern.
- **Local control:** The degree of modification to each subject when running an experiment is called local control. Local control can be a serious problem in software engineering research, because the project management will not allow a disturbance in a software development project adding additional costs and risk.
- **Influence of context:** In software development, it can be hard to determine all the factors that influence the results in an experiment. Usually a lot of people, tools, methods, and artifacts are involved in software development projects, all affecting the experiment. One example is that in most software companies, the software developers have un-documented procedures which they follow when developing software, that can be totally different from the documented, formal procedures that *should* be used.
- **Temporal properties:** Data collection may be historical or current. Historical data may be missing the information needed to come to a conclusion.

When we did an evaluation of our CAGIS PCE where we compared it to two other process centred environments (presented in chapter 20) we had to consider how data were collected. Although the data we collected in this experiment were limited, we addressed the problems with collecting data described above: First, it was important that our experiment could be replicated by other researchers by defining the experiment in enough detail and clarity. Then we identified the two attributes we wanted to investigate: 1)

How much of the scenario could be modelled and supported (*coverage*), and 2) How well could the prototypes cope with specific process changes (*adaptability*). The experiment further described in detail how these attributes should be measured using the GQM method. Local control and Influence of context were major problems in our experiment, since the three process centred environments were not used in a real software engineering environment. When evaluating the results we have taken these two factors into account, and used additional qualitative reasoning to produce a valid result. Temporal properties have not been a problem for our experiment.

A fundamental problem concerning the results from an experiment is to determine how valid the result is. Even if the data have been collected correctly, it does not automatically mean that the result is generally valid. Internal validity means that the result of an experiment is valid only within an organisation the experiment was designed for, while external validity means that the result is applicable outside the scope of the experiment. For software engineering experiments it is very hard to get results that are external valid, because there are so many factors that can change the result of an experiment such as the size of the organisation, how the company is organised, what products are being made, how the development process is organised etc. The results from evaluating our CAGIS PCE approach can not be said to be external valid, since it is not validated in several real environment.

3.2.3 Models for Validation Technology

Zelkowitz and Wallace have developed a taxonomy for software engineering experimentation based on various examples of technology validation [ZW98]. This taxonomy describes 12 different experimental approaches that are grouped into three broad categories: Observational methods, historical methods, and controlled methods.

Observational Methods

The observational method collects relevant data as a project develops, and is characterised with little control over the development process other than using the new technology being studied. There are four types of observational methods:

1. **Project Monitoring** is collecting and storing data during a project without interfering with the project. Project Monitoring can be used to get a status quo of how software is currently developed.

2. **Case Study** is an experiment where a certain attribute is monitored and data are collected to measure that attribute, in order to investigate a specific goal for the project. Data are often collected by people involved in the projects through interviews or forms.

3. **Assertion** is an ad-hoc validation method where developers execute their own experiment to see if one proposed technology is better than other alternatives.

4. **Field Study** is a validation method where often several projects are simultaneously studied by an external group so the subject under study is not disturbed.

Historical methods

When historical methods are used, existing data are collected from completed projects. Historical methods can be divided into four methods:

5. **Literature search** analyses results of papers and other documents to confirm an existing hypothesis or to improve the data collected in one project, with more similar data.

6. **Legacy data** studies previously completed projects by collecting all available quantitative data. Data that can be interesting to look at can be products, design documents, source code, test documentation, how the project was organised etc.

7. **Lessons learned** studies qualitative aspects of previous projects to improve future projects by typically interviewing project members about impact on the introduction of new technology.

8. **Static analysis** is a validation method focusing on collecting and analysing quantitative data related to completed software products. Typical data that can be measured is lines of code, number of modules, module dependencies etc.

Controlled methods

The controlled method is the same research method as used in experimental design in other scientific disciplines. If sufficient instances of an observation are available, this research method provides statistical validity of the result. There are four types of controlled methods:

9. **Replicated experiment** is an experiment where several subjects are set to perform a task in multiple ways. The researchers control the experiment, by setting control variables such as method used, tools used, staff used etc.

10. **Synthetic environment experiments** are experiments where the execution environment is changed from its original setting to make the experiment more handable to perform. In such experiments, the organisation is often scaled down, and only parts of the software development process are executed focusing on specific issues.

11. **Dynamic analysis** is a way of evaluating products when they are executed, by adding source code in the products making it possible to debug and measure performance when the products are executed. By using this method, it is possible to compare how efficient similar products execute similar functionality.

12. **Simulation** is a method where a model of the real environment is used in the experiment of validating technology. The model is applied to some products (executed), and data are collected from this simulation.

In this thesis we have used two of the research methods described in the list above. First, we have used the **assertion** method to validate if our approach is better than other existing approaches. Second, we have conducted a **literature search** in the beginning of our project to see what we could learn and use from a previous project. More details about the research methods we used are given in section 3.3.

Zelkowitz and Wallace performed a literature study in [ZW98] where they examined what validation methods were used in a selection of software engineering research papers (IEEE Transaction on Software Engineering, IEEE Software, and proceedings from International Conference on Software Engineering). They discovered that about 30 % of the papers did not have any experiments at all, and about that 34 % of the papers used *Assertion* for validation. For the rest of the papers *Case study* (10 %), *Lessons learned* (9 %), and *Simulation* (5.5 %) were the most popular methods used. It was also discovered that the number of papers using different validation methods in e.g., IEEE Transaction on Software Engineering do not differ much from numbers found for so-called hard sciences.

These were methods geared towards validation of technology (tools, architectures, models etc.) for software engineering. A persistent problem in software engineering research is to validate the proposed technology. The reason for this can be the large costs and risks with running full-scale experiments in real environments for validation purposes. The number one priority for many software companies is to produce software fast. Validation of new technology can add work-load to software developers, and is therefore undesirable. It is possible to reduce additional costs and risks by using research methods, like lessons learned, simulation, and synthetic experiments that do not affect on-going projects. However, the results are not automatically valid in the real developing environment. According to Zelkowitz and Wallace's literature study, almost one third of the investigated papers did not have any experimentation at all, and another third used *assertion* as the research method.

3.3 Research Methods used in this Thesis

The research methods used in this thesis can be divided into three main parts: A **lessons learned** study to provide the initial requirements for our approach, an **internal validation** of our prototypes in order to enhance and improve their functionality, and a final validation of our CAGIS PCE using the **assertion method**.

3.3.1 Lessons Learned from the EPOS Project

Initially, the work with this thesis started with a *lessons learned* study of our own EPOS process environment. From studying the EPOS project, we discovered that the implementation of EPOS suffered from being too centralised, too many tightly coupled components, based on out-dated technology, and running only on very specific hardware and software configurations. To avoid falling into the same ”traps” as we did in EPOS, we conducted a thorough prestudy in order to provide the necessary technology and architectures. Based on the experiences from EPOS and the prestudy, we decided that the CAGIS PCE prototype should consist of loosely coupled tools that can be distributed and open to other tools, and running on various hardware and software configurations. The EPOS project experiences on how to model processes and how to deal with evolution of software processes (described in the background papers in the chapters 8 and 9) were used as basis for process modelling in CAGIS. Further, the work on cooperative support in EPOS (chapter 10) providing cooperative awareness services was extended to a cooperative multi-agent architecture used in the CAGIS project. The lessons learned study gave us some warnings about what we should not do in the CAGIS project, as well as useful experience and knowledge that have been further explored.

3.3.2 Internal Validation of the CAGIS PCE

During building the CAGIS PCE, we have validated our implementation internally to investigate if the different components can and will do what they are supposed to. Through simple, small scenarios illustrating critical issues in cooperative software engineering, we have tested that our prototypes provide the specified functionality, such as to provide support for doing efficient resource negotiation, for flexible re-allocation of activities, etc. Based on these functional validations we have initiated new implementations of the prototypes to enhance the existing functionality. In addition to validating the functionality of the prototypes, we have also evaluated the technology used to implement the prototypes in regards to the functionality offered and future support from the technology providers. The technology evaluation of our agent architecture forced us to change the underlying agent architecture from Aglets (IBM) to JavaSpaces (Sun). The reason for this change of technology was that Aglets was no longer supported by IBM, Aglets did not support newer versions of Java (Java 2 or later), and JavaSpaces offered better support for changing our agent system in run-time through JINI.

3.3.3 Validation of our Approach using the Assertion Method

The final validation of our work was an evaluation performed by the *assertion method*. Our technology, as well as two other SPT technologies, were evaluated by modelling the same conference organising scenario using all three SPT technologies and by comparing the results. We have also shown that our technology can model other scenarios.

The comparison between the three technologies was based on limited measurements of process model completeness and adaptability. In addition, a discussion based on the measurements was used to highlight strengths and weaknesses of the mentioned technologies. The developers of the other technologies used in this evaluation, have themselves suggested how to model a conference organising scenario, and also performed some modelling themselves. The assertion method was chosen as research method for the final validation because it was impossible to do “real validation” in a software company within the allocated budget and time.

3.3.4 Comments

In this thesis we have chosen to validate our CAGIS PCE prototype from a technical point-of-view. We have tested that our system has the functionality needed to support cooperative software engineering processes, and how our system technically performs. Another approach would be to validate our system from the user’s point-of-view, and investigate how our system helps users cooperate in a working environment. Research within SPT usually has focused mainly on technical issues, while CSCW research has looked more into the human aspects. A complete validation should cover both technical and human issues, but we did not have time to do this in this thesis.

CHAPTER 4

Own Contribution

4.1 Background for this Thesis

The work with the CAGIS Process Centred Environment has its root in the EPOS PSEE, also developed at our department. EPOS consisted of two main parts: EPOS CM taking care of software products through advanced transaction handling, and EPOS PM offering a rich toolset to model and enact process models. In EPOS, we wanted to enhance its cooperative support and its ability to change enacted process models on-the-fly. We also looked at possibilities for making EPOS support distributed work, making user-clients available on the Web. Over time EPOS had grown into a very large system, making it almost impossible to maintain. A large number of students, researchers and guest researchers had been programming many lines of Prolog and C. The different contributions to the system were not all closely integrated, and it became a big problem to make the EPOS PSEE work as an integrated environment.

The cooperative support added to EPOS [WLCM98b] offered a light-weight awareness support (i.e. notification) for dealing with update access conflicts of artifacts. Cooperative resolutions of access conflicts were offered through flexible locking mechanism, awareness of access conflicts, synchronisation mechanisms for files, and merging of files where two or more users had changed the same file.

To give on-the-fly support for changing an enacted process model, a set of operations were offered in the graphical task-network browser [NWC97]. These operations made it possible to directly manipulate the task-network by adding activities, remove activities, manipulate the sequence of activities, and change attributes to activities and artifacts.

Although we managed to add cooperative support to EPOS and some support for on-the-fly process evolution, the different parts of EPOS were almost impossible to integrate. The EPOS prototype suffered from typical legacy system problems: undocumented code and features, developers that knew the system started to work for other institutions (commercial companies), and it was hard to update the system to newer versions of operating system and newer versions of programming language and programming packages.

When the CAGIS projects started in 1997, we wanted to focus on cooperative process support, distribution of the process, and on support for on-the-fly process evolution. Based on the experiences with EPOS, we wanted the prototypes in CAGIS to consist of a set of rather small tools that were relatively easy to maintain and were loosely coupled. Another important aspect of the CAGIS prototype was to use recent technology to allow web-integration, open architecture accessible to other systems, as well as extensive distributed support for management of processes and artifacts. More details about the architecture of our prototype are described in section 4.3

4.2 System Requirements

In section 1.3, research questions for this thesis were presented. These research questions identify the topics of concern. The system requirements have been worked out based on the research questions as a starting point to list requirements to fulfil the implementation of the CAGIS PCE. The requirements are divided into two main parts: non-functional and high-level functional requirements.

4.2.1 Non-Functional Requirements

The non-functional requirements describes what properties the CAGIS PCE should have that are not directly related to functionality offered, and can be divided into five parts:

N1 **Openness**
Based on experiences from the EPOS project as mentioned in section 4.1, the CAGIS PCE prototype should be open to existing tools and systems, and it should run various hardware and software platforms. By open to existing tools and systems, we mean that the prototype should be easy to integrate with existing software components and tools, and standard interfaces should be used to communicate to other system whenever possible.

N2 **Software and Hardware Requirements**
It should be possible to run our prototype at a computer matching the power of a Pentium 120 MHz or better, and with 32 MB Ram. The computer must have software and hardware installed for accessing the Internet.

N3 **Response Time**
The CAGIS PCE prototype should prioritise functionality before performance, but users of the prototype should not wait for more than 30 seconds for reactions on user interactions at normal load of the network.

N4 **Security**
The CAGIS PCE prototype does not provide any additional security than the security provided by the underlying technologies used.

N5 **Maintainability and Extensibility**
The CAGIS PCE prototype should be easy to configure, maintain and expand. Rather than implementing one big system the prototype should consist of several loosely coupled tools, and it must be possible to add more tools later on.

4.2.2 High-level Requirements

One of the initial objectives for the CAGIS project [C+96] was: "*To give cooperating human problem-solvers (designers, engineers) better support for concurrent and distributed team work.*". Based on this CAGIS project objective, a number of research questions were identified as described in section 1.3. These research questions can be summarised as following: Investigate what is needed to model and enact dynamic distributed, cooperative and individual processes in a heterogeneous environment, and investigate how to create an infrastructure for doing so. Based on these research questions, we have identified seven high-level functional requirements the CAGIS PCE should implement:

HF1 **Local work support**
The CAGIS PCE should provide tool support for guiding users through their own individual working processes by telling the user what to do, and by making tools and documents accessible for her/him.

HF2 **Cooperative work support**
The CAGIS PCE should support cooperative work processes, for coordinating object, negotiating about shared objects or resources, delegating work, voting on proposals etc.

HF3 **Distributed work support**
The CAGIS PCE should support processes that are geographically distributed.

HF4 **Light-weight support**
The CAGIS PCE should use process concepts and formalisms that are easy to understand and apply for the user.

HF5 **Dynamic support**
The CAGIS PCE should provide an infrastructure that can cope with changes of the process, and is easy to configure and re-configure.

HF6 **Local control**
The CAGIS PCE should allow users to define and change their own local processes.

HF7 **Managerial control**
The CAGIS PCE should allow management, that represents users involved in cooperative work processes, define and change cooperative rules for interaction.

In section 2.3.2 in the State-of-the-art chapter, the two main research challenges for Process-centred Software Engineering Environments (PSEEs) were identified to be how to provide proper cooperative process support (motivated by the creative activities in software development), and how to offer flexible process support (because software processes are highly unstable). Traditional PSEEs have not been able to address these challenges, because they have focused on providing strict process support provided by centralised, inflexible architectures. In our work with the CAGIS PCE, we have focused on cooperative support and flexibility which is reflected in the functional requirements above (HF1-HF7).

4.3 Architecture

This section describes the CAGIS PCE architecture and its components. The first part of this section presents a motivating scenario used in the rest of this section. Then the overall CAGIS PCE architecture is presented. The sections 4.3.3 to 4.3.5 present the three main components of the CAGIS PCE architecture: The CAGIS DIAS, the CAGIS SimpleProcess, and the CAGIS GlueServer. The last section (section 4.3.6) describes how the three main components of our CAGIS PCE architecture were designed.

4.3.1 A Motivating Scenario

To make it easier to see how the different components in the CAGIS PCE interacts, we will present a small scenario. The scenario describes a part of a process in a computer game company named CoolGames, where we focus on two departments; the software development department and the graphical design department located in Oslo in Norway and in San Francisco in the USA respectively. Since these departments are geographically distributed, they use computers to interact via the Internet. Figure 4.1 illustrates a part of the development process for a new game for the two departments. In CoolGames' new project, they have some initial ideas of their new 3D graphics game.

The developers start to implement a 3D graphics-engine prototype (A1) while the designers are drawing some concept-drawings (B1). After some time of development and designing, they decide to have some brain-storming (C1) where they share ideas, drawings, and testing results from experiments with the 3D graphics-engine. Based on the result from this brainstorming, the process could go back to either one of the activities A1 or A2, or both, or to continue to the activities A2 and B2. In the activities A2 and B2, each department estimates how much human resources they will need on their own

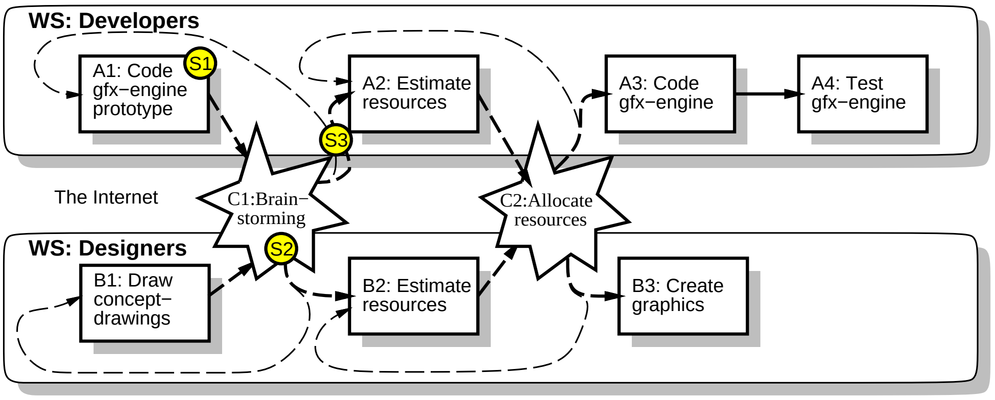


Figure 4.1: A scenario used to illustrate the CAGIS PCE architecture

and how much resources they need to acquire from the other department. A negotiation is then initiated (C2) between the two departments, where they negotiate about how much resources they should get. If the allocation process goes into a deadlock, the two departments must change their estimates (go back to A2 and B2). After a successful allocation, the process can proceed to the activities A3 and B3. The activity A4 will be started as soon as the activity A3 is finished.

In figure 4.1 we can divide the process into three parts:

S1 **Individual activities** are activities that can be performed by individuals without any interaction with other persons. The activities A1-A4 and B1-B3 are individual activities.

S2 **Cooperative activities** are activities that only can be performed when more than one person is involved. The activities C1 and C2 are cooperative activities.

S3 **Cooperative rules** are relations between cooperative and individual activities (e.g. between C1 and A2) drawn in the figure as dotted lines.

The classification above will be used later in this section to describe how the different parts of the CAGIS PCE interact and what kind of activity support they provide.

4.3.2 The CAGIS PCE Architecture

In the CAGIS project, we wanted to look at how Cooperative Software Engineering (CSE) could be supported. By Cooperative Software Engineering (CSE) we mean large-scale software development and maintenance work which falls into the two categories *coordinated* and *cooperative workflow* (see section 2.5.1). Because of the rapid spread of World Wide Web as the standard underlying platform for CSCW systems and other systems, more software companies are moving from the traditional centralised working style to the decentralised one. In decentralised CSE, communication, coordination, collaboration, and negotiation among the various participants are more complicated, because people are not only geographically distributed, but may also work on different platforms, at different times, with different process models.

The key issues of CSE are group awareness, concurrency control, communication and coordination within the group, shared information space and the support of a heterogeneous, open environment which integrates existing, single-user applications. All these are related to the software process.

Nowadays, it is believed that the Multi-Agent Systems (MAS) offer a better way to model and support these distributed, open-ended systems and environments [CMM97, CM96]. A MAS is a loosely-coupled network of problem solvers (agents) that work together to solve a given problem. The main advantages of a MAS are: *Decentralisation* where complex systems are broken down to cooperative subsystems, *Reuse* of previous components, *Cooperative Work Support* to better model and support the spectrum of interactions in cooperative work, and *Flexibility* to cope with incomplete specification, constant evolution, *Scalability* to distribute computation on several computers, and open-endness to other systems. These advantages made us choose to use a MAS as a central component of our architecture, because it was ideal for implementing cooperative support in a distributed environment. By using a MAS to implement cooperative support it was required to program agents that could provide this support using an agent API. For simple individual activities we found that could be a over-kill, because it would take too much time to program these simple processes. Therefore we then chose to add a workflow component into our architecture that was specialised on modelling and enacting simple local workflow processes. By keeping the workflow process modelling language simple, it was possible for users to model and change their own processes. Our architecture now consisted of a MAS and a workflow system that were not connected in any way. Thus, we needed a middleware to glue the MAS and the workflow tool. By allowing the workflow tool and the MAS to be loosely coupled through the middleware component, we could also allow other agent and workflow systems to interact in one heterogeneous environment.

We have chosen an architecture consisting of these three main components also shown in figure 4.2 providing process support according to the classification described in the last part of section 4.3.1:

S1 The **CAGIS SimpleProcess** workflow tool provides local process support for *individual activities*. If we recapitulate to the scenario in figure 4.3.1, we can identify the activities A1-A3 and B1-B3 as activities modelled and supported by the CAGIS SimpleProcess tool.

S2 The **CAGIS Distributed Intelligent Agent System** provides support for *cooperative activities* involving people working in a distributed environment. The activities C1 and C2 involve more than one role and spans across workspaces (distributed), and are regarded as cooperative activities supported by the CAGIS DIAS.

S3 The **CAGIS GlueServer** makes it possible for the CAGIS SimpleProcess workflow tool to interact with the software agents by specifying the *cooperative rules* between individual workflow and cooperative workflow in a GlueModel. The relations between individual and cooperative activities (represented in figure 4.3.1 as dotted lines) indicate the cooperative rules that are modelled in the GlueModel.

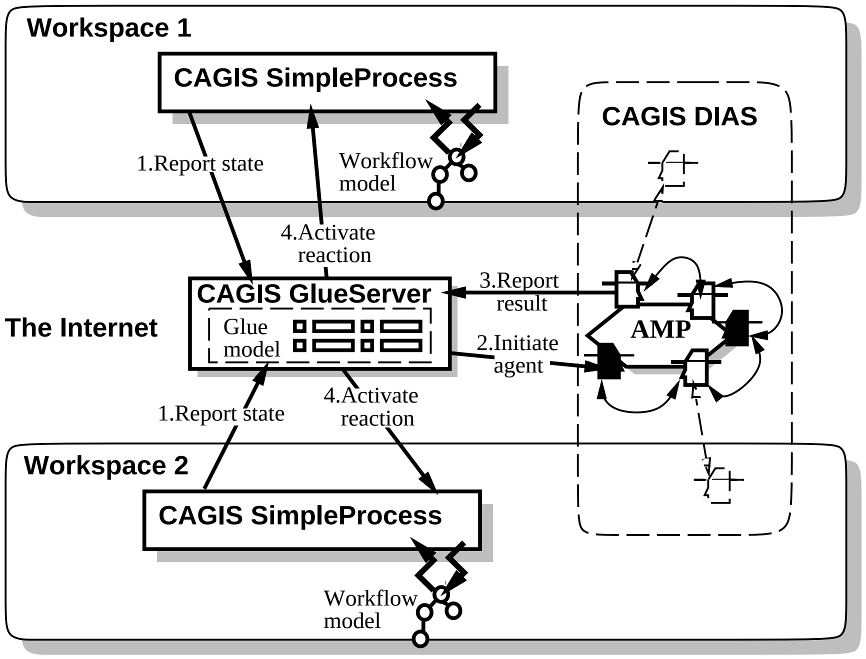


Figure 4.2: Architecture for the CAGIS PCE

A typical interaction between the different components in the CAGIS PCE will work according to the four steps as shown in figure 4.2:

1. The CAGIS SimpleProcess workflow tool will *report its state* to the CAGIS GlueServer, e.g. that it is finished with executing the activity A1 in the motivating scenario.
2. The CAGIS GlueServer will look through the GlueModel to see if anything is specified for the activities A1 abd B1, and it will *initiate brain-storming agents* in the CAGIS DIAS (the cooperative activity C1).
3. The brain-storming agent can return two results: successful or unsuccessful. The *result is reported* to the GlueServer when C1 as finished.
4. The GlueModel specify different reactions depending on the result reported to the GlueServer (successful or unsuccessful). Depending on the result, the GlueServer

will *activate a reaction* in the CAGIS SimpleProcess, the CAGIS DIAS or the CAGIS GlueServer. In our scenario a successful result from the cooperative activity C1 will activate a reaction by the CAGIS GlueServer to execute the activities A2 and B2 in the CAGIS SimpleProcess workflow tool. An unsuccessful result will activate the CAGIS SimpleProcess workflow tool to re-execute the activities A1 and B1.

The CAGIS SimpleProcess and the CAGIS GlueServer communicate through CGI, and the CAGIS DIAS and the CAGIS GlueServer communicate through CORBA using the MASIF [1] standard. The CAGIS PCE architecture is flexible since the CAGIS GlueServer can be used to interact with other agent systems through the mobile agent interface, and the CAGIS GlueServer can also be used to communicate with other workflow tools through the interoperability workflow-XML binding framework. In this way, the CAGIS PCE can federate systems offering a variety of process support. Note that software agents can interact both with the CAGIS GlueServer and directly with the users in their workspaces.

In addition to the three main parts of the architecture, we have outlined a design of how software agents can be used combined with an existing CAGIS Process Centred Environment to deal with evolution of distributed, fragmented workflow models. This design proposes a solution to solve consistency problems when process models are changed in the workflow tool. We propose to use mobile software agents, offering awareness services solving conflicting updates of process fragment. More details are described in paper 12 (section 19).

The architecture of the three main components of the CAGIS PCE will be described in the following sections.

4.3.3 The CAGIS DIAS Architecture

Our multi-agent architecture is an extension and specialisation of the more general **Agora** architecture proposed by Matskin et al. [MDP98] suitable for modelling and supporting all kinds of cooperative work. The CAGIS DIAS architecture is a framework for implementing support for communication, coordination, collaboration, and negotiation among the various participants grouped in workspaces in a CSE process. The main components in this architecture are agents, workspaces, AgentMeetingPlaces(AMPs), and repositories.

The non-functional requirements (N1 and N5) described in section 4.2.1 were the starting point for our multi-agent architecture. We wanted the CAGIS DIAS to be easy to configure and expand, and use free standard software components whenever available. We also wanted to use mobile agents, because they provide efficient usage of network bandwidth, and less computation on the server is needed. Figure 4.3 shows the initial

[1] MASIF is short for Mobile Agent System Interoperability Facility defined by the Object Management Group (OMG).

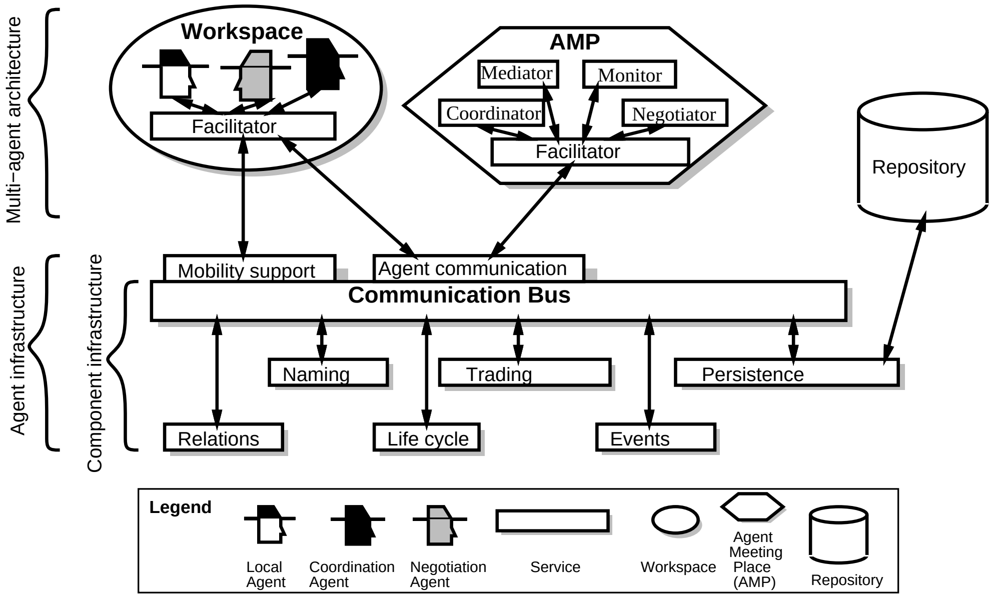


Figure 4.3: Design of the CAGIS DIAS architecture

design for our multi-agent architecture. We have used a multi-tier architecture based on the agent, places, and things paradigm. The lower part of the figure (component infrastructure and agent infrastructure) defines the foundation, based on available standard implementations, which will provide functionality and services to the prototype of the multi-agent architecture. Typical CORBA services are identified in the component infrastructure such as naming, relations, life cycle, trading, events and persistence. A central component both in workspaces and AMPs in our architecture is the facilitator that simplifies the implementation of agent communication, agent security, the mediation between agents, and the monitoring of agents. The reason for this is that the interaction between various entities can be controlled from one central point. The drawback with this solution is that the facilitators might become bottlenecks of the system. All communication between the components in the architecture is provided through the communication bus. Agent specific services for mobility and agent communication are also provided as services connected to the communication bus. An AMP is in this figure described as a server providing various services through a facilitator. Repository support is provided through a persistence service.

Figure 4.4 illustrates the recommended technologies that should be used for the various parts of the CAGIS DIAS architecture based on a technology study described in [Øye98]. The technology study suggested to use *Java and Java IDL* as the component infrastructure because Java provides code portability, Java is a broadly accepted standard, many agent-related technologies are implemented in Java, and Java is updated frequently and available for free. By choosing Java IDL as a CORBA implementation, no other CORBA services

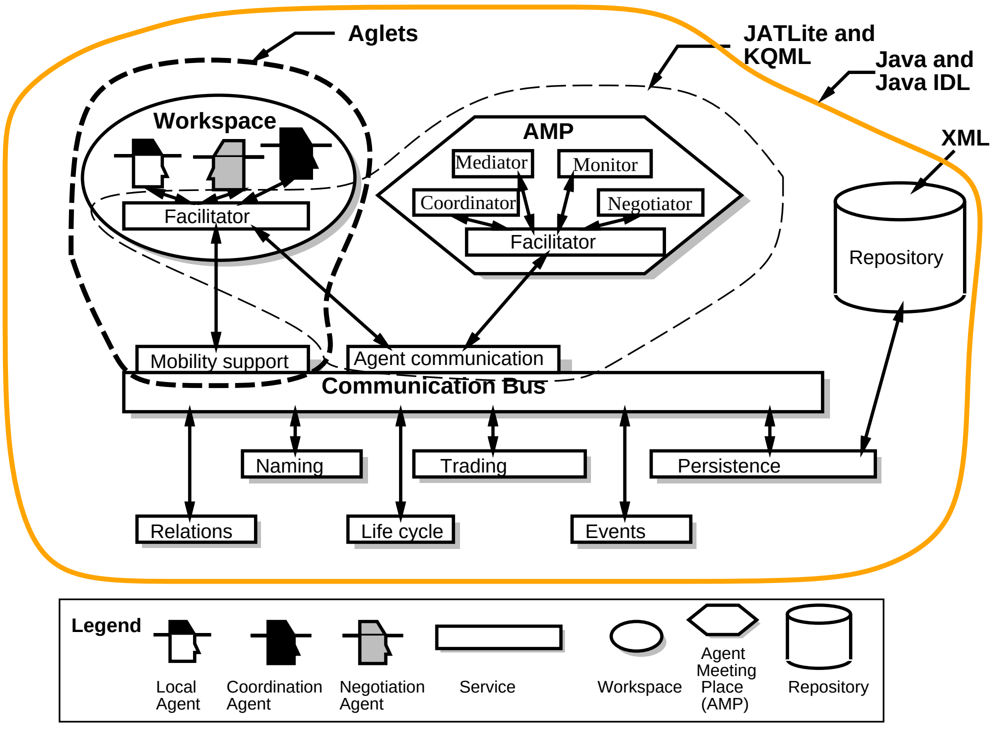


Figure 4.4: Recommended technologies for the CAGIS DIAS architecture

than the naming service were supported. The other services must then be developed in Java or can be replaced if Java IDL will offer them in the future.

To provide support for agent communication and as a foundation for implementing AMPs and workspaces, we suggested to use *KQML and JATLite*. We recommend to use KQML [FFMM97] as the agent communication language, because it is an extensible standard that has many features required for an agent communication language. JATLite is a Java implementation providing inter-agent communication through KQML and facilitates the administration of, and communication between a group of related agents.

Further to provide mobility support for agents, the *Aglets framework* from IBM [LO98] was selected. The Aglets framework was chosen because at the time we conducted the technology study, the Aglets implementation was closest to OMG's Mobile Agent Facility specification. In addition we suggested to use *XML* to represent information and work-productions in the architecture because a lot of XML tools are available in Java, and XML does not put any restrictions on the format of the information it shall represent.

It should be noted that the lines drawn in figure 4.4 only loosely denote where the various technologies should be used, and more experience is needed to decide exactly which of the technologies are best in the specific situations. The actual implementation of the CAGIS

DIAS is described in section 4.4.1. This architecture is also described in paper 7 (section 14), paper 8 (section 15), and paper 9 (section 16).

4.3.4 The CAGIS SimpleProcess Architecture

In individual processes where communication, negotiation, cooperation or collaboration between participants in workspaces are not required; a simple activity-based workflow model is sufficient. By using a workflow tool to take care of such processes, it is simpler for the participants to model and change their own processes, since no programming is required. Flexibility has been the main motivation when designing the CAGIS SimpleProcess, by allowing the process model to be re-arranged and changed during enactment. This flexibility allows us to gradually build a process model from existing process fragments, to allow parts of the process to be unspecified, and to re-arrange and change the sequence of activities of the process model run-time.

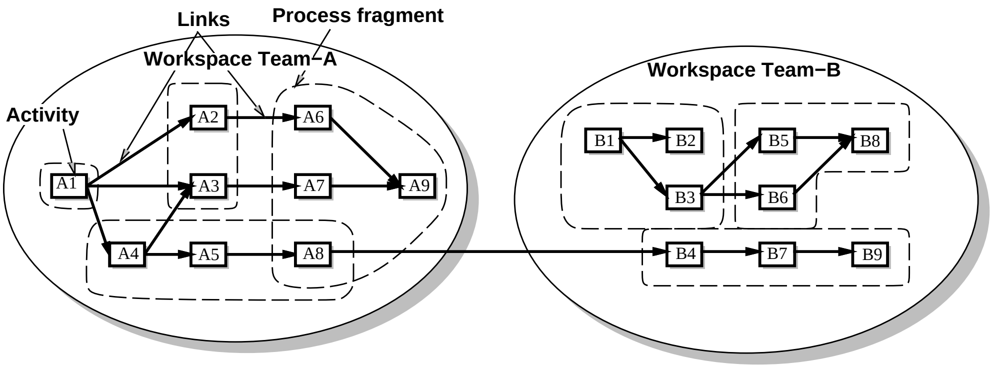


Figure 4.5: CAGIS SimpleProcess concepts

In the CAGIS SimpleProcess workflow tool, a process is represented as shown in figure 4.5. A process model can consist of several autonomous parts called **process fragments**. A process fragment has a name and is associated with a workspace and can consist of one or more **activities** located in private or shared workspaces. Relationships between activities are called **links** that define the execution sequence of the activities. A **workspace** is identified by an URL that must be accessible for the CAGIS SimpleProcess. Process models in CAGIS SimpleProcess are specified in XML and the document type declaration (DTD) of the CAGIS SimpleProcess PML is as shown in figure 4.6. Since we have used a XML DTD to specify our PML, it is possible to change the PML if required e.g. by adding new part. The figure shows that an activity has a name and is located in a workspace and is defined by the parts: Prelink(s), postlink(s), a state, a due time, a feedback option, a description, and a code part. The *code* part is used to specify HTML-code or an URL to a web-page to be presented when an activity is activated. The HTML-code can be used to to present some texts and pictures, list hyper-links to important documents and tools, present the user HTML-forms, or executing Java-applets. Prelinks specify the activities to be executed before current activity, and postlinks specify the activities to be executed after current activity. An activity will go from the state *Waiting* to *Ready*, if all prelinks (the prior activities) have the state *Finish*. However, if the activity is specified with a feedback loop, only one of the prelinks needs to have the state *Finish* to be activated. The user must press a button to explicitly declare that she/he is finished with an activity. The activity network works similar to hyper-linked web-pages with states. Figure 4.7 shows the XML-code for specifying the pre- and postlinks for the activity A3 shown in figure 4.5.

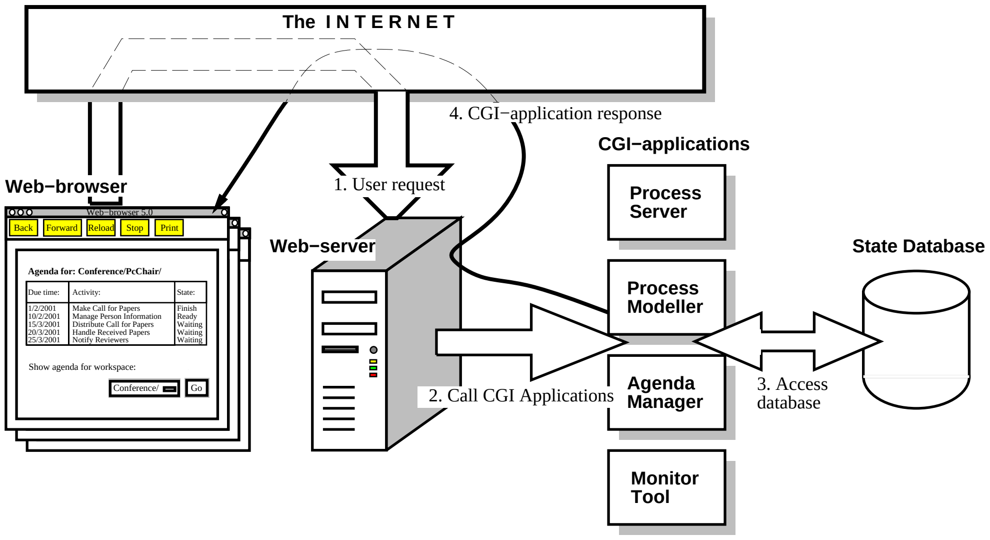
```
<?XML encoding=''UTF-9''?>
<!ELEMENT process (name,
                  (processfragment)+>
<!ELEMENT processfragment (name,
                          (workspace),
                          (activity)+)>
<!ELEMENT activity (name,
                   (workspace),
                   (prelink)*,
     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

Figure 4.6: XML Document Type Declaration of the CAGIS SimpleProcess PML

```
<Activity>
  <Name>A3</Name>
  <Workspace>Team-A</Workspace>
  <Prelink>Team-A/A1</Prelink>
  <Prelink>Team-A/A4</Prelink>
  <Postlink>Team-A/A7</Postlink>
  ...
</Activity>
```

Figure 4.7: An example of use of $< prelink >$ and $< postlink >$ tags

Figure 4.8 shows the architecture for the CAGIS SimpleProcess. The figure shows how the user interacts with the workflow system through four steps:



Agenda for: Conference/PcChair/

Due time:	Activity:	State:
1/2/2001	Make Call for Papers	Finish
10/2/2001	Manage Person Information	Ready
15/3/2001	Distribute Call for Papers	Waiting
20/3/2001	Handle Received Papers	Waiting
25/3/2001	Notify Reviewers	Waiting

Figure 4.8: CAGIS SimpleProcess architecture

1. The user sends a **user request** from the web-browser through the Internet to the Web-server running CAGIS SimpleProcess. This user request is specified through a HTML-form providing the user-interface for CAGIS SimpleProcess. Typical user requests can be to look at the agenda for a specific workspace, activate an activity, notify that an activity is finished etc.

2. The web-server will then **call CGI-applications** according to the user request. CAGIS SimpleProcess consists of four CGI-applications: A process server responsible for managing process changes and process states, a process modeller providing an easy way for users to enter process models through a web-interface, an agenda manager presenting user agendas and activities, and a monitor tool providing a user-interface to monitor the process.

3. The active CGI-application **access database** for process states and process information (both read and write).

4. A **CGI-application response** is sent back to the web-browser as HTML.

In addition to the steps shown in figure 4.8, the CGI-applications access XML-documents representing the process model in the users workspaces. Distributed workspace technology like BSCW [BHT97] and WebDAV [Whi97] can be used to provide workspace support. The workspace support provided in the CAGIS environment has not been the focus of this thesis, and is covered by other initiatives in the CAGIS project.

A more detailed description of this workflow tool is described in paper 10 (section 17).

4.3.5 The CAGIS GlueServer Architecture

The *GlueServer* is a piece of middleware used to provide interaction between multi-agent systems (CAGIS DIAS) and the workflow systems (CAGIS SimpleProcess). A *GlueModel* specifies the relationship between process fragments and agents, making it possible for process fragments to delegate tasks to software agents, to use software agents to evaluate what to do next after completion of execution of a process fragment, or to monitor the environment for events to detect exceptions. By using the GlueServer to combine agent systems with workflow systems we can achieve better support for modelling and enacting cooperative activities and individual activities respectively. In addition, the GlueServer can be used to federate several agent systems and workflow systems into one heterogeneous, loosely coupled process centred environment.

A GlueModel is specified in XML using the *Glue Modelling Language*. Figure 4.9 shows the GlueModel for modelling the dependencies between the activities A1 and C1 in the scenario presented in figure 4.1 in section 4.3.1. The GlueModel specifies that as soon as the workflow tool reports that the process fragment “A1:Code gfx-engine” is finished, a brain-storming agent should be initiated in the agent system representing the activity “C1:Brain-storming”. Based on the result returned by this brain-storming agent (successful or unsuccessful), the workflow tool should execute the process fragment A2 or A1 respectively.

Here is a more detailed explanation of the GlueModel. The *first part* of the GlueModel specifies the **agent** involved in the cooperative activity by an agent class and an amp-id to the agent place where the the agent will interact with other agents (CoolGamesAMP). The **interaction type** specifies how the workflow system and the agent system should interact. When *Periodic invocation* is used, agents decide what to do next at the termination of a process fragment. There are two other interaction types:

- *Predefined interface* meaning that the workflow tool delegates an activity to an agent.
- *Dynamic monitoring* where monitoring agents are continuously probing the environments for certain events or states. Whenever an abnormal situation is detected by an agent, this abnormal situation is reported to the GlueServer that can execute a reaction in the workflow tool (e.g. to change a process fragment, halt a process fragment, execute a specified process fragment etc.)

The **result** tag is used to specify what values the agent can return.

The *second part* of the GlueModel specifies the **process fragment** by a process fragment ID. The rest of the process fragment part is used to describe a **reaction** specified by **result** - **action** pairs. The result is the possible results returned from the agent, where as the action specifies what to do if there is a match. The predefined actions in the Glue modelling language are: execute process fragment, move process fragment, halt process fragment, change and re-execute process fragment, add new process fragment, remove process fragment, start new interaction (agent system), stop interaction (agent system), create a new-AMP (agent system), remove AMP (agent system), and change fragment agent pair (GlueServer)[2].

```
<fragment-agent-pair>
  <agent agent-class="agents.brainstorming" amp-id="CoolGamesAMP">
    <interaction-type>Periodic invocation</interaction-type>
    <result>successful|unsuccessful</result>
  </agent>
  <fragment fragment-id="Developers/A1:Code gfx-engine prototype">
    <reaction>
      <result>successful</result>
      <action fragment-id="Developers/A2:Estimate resources"
              body="execute_process_fragment_PFNUMBER"></action>
      <result>unsuccessful</result>
        <action fragment-id="Developers/A1:Code gfx-engine prototype"
              body="reexecute_process_fragment_PFNUMBER">
        </action>
    </reaction>
  </fragment>
</fragment-agent-pair>
```

Figure 4.9: CAGIS GlueModel example

The user should be able to specify the GlueModel directly in XML, or use a tool with a graphical user-interface for entering the information required.

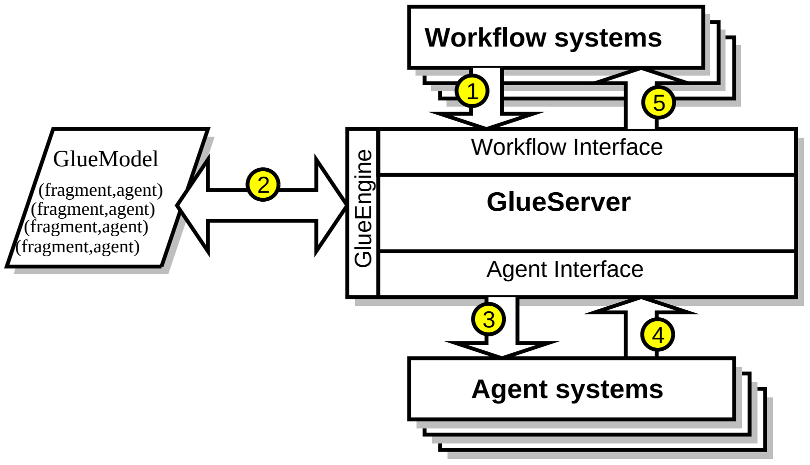


Figure 4.10: CAGIS GlueServer architecture

Figure 4.10 shows the architecture of the CAGIS GlueServer consisting of three main components:

[2]The actions without parenthesis are executed in the workflow tool.

- **GlueEngine:** The main purpose of the GlueEngine is to parse the GlueModel and look for process fragment - agent pairs in the model matching with state information received from the workflow System or the agent system. If a process fragment - agent pair is found, the GlueEngine will initiate a reaction through the workflow interface (typically move process fragments, re-execute process fragments etc.) or the agent interface (initiate negotiation agent, create a new agent meeting place etc.).

- **Workflow Interface:** The workflow interface interacts with workflow systems through XML-interface.

- **Agent Interface:** The agent interface interacts with agent systems through a MASIF interface.

A typical interaction between the three components in the CAGIS PCE can be as follows (the numbers are illustrated in figure 4.10):

1. The Workflow system reports its state to the GlueServer via the workflow interface.

2. The GlueServer finds a process fragment - agent match in the GlueModel using the GlueEngine.

3. The agent interface initiates an agent as specified in the GlueModel.

4. The agent system reports the result of an agent interaction back to the GlueServer through the agent interface.

5. The GlueServer will activate a reaction according to the GlueModel.

The reaction can activate a specified operation in the workflow tool, but it can activate an operation in the GlueServer or in the agent system. To enable other workflow tools to interact with the rest of the CAGIS PCE, the GlueServer architecture should offer an interoperability workflow-XML binding specified by the Workflow Management Coalition. With the interoperability workflow-XML binding, the GlueServer can interact with other workflow systems that support this feature. The GlueServer is described in more detail in paper 11 (section 18).

4.3.6 Detailing the Architecture

Based on the high-level requirements described in section 4.2.2 and architecture of the CAGIS PCE described above, we will now describe how our architecture is designed in more detail. A set of design specifications will identify the different parts of the prototype that should be implemented, described below for the three main parts of the CAGIS PCE prototype.

The CAGIS DIAS Design Specification

The CAGIS Distributed Intelligent Agent System (DIAS) should provide a framework for supporting cooperative activities between people in different workspaces and people in the same workspace. Here is a summary of the full list of design specifications (D1-D43) described in Appendix A:

DD1 **Agent specifications:**
An agent should register itself in an Agent Meeting Place (AMP) according to the agent properties defined in MASIF standard. The CAGIS DIAS should provide three main types of agents:

- *System agents* are responsible for creating, deleting, and managing AMPs. System agents are also responsible for monitoring agent activity, accessing repositories and external agent systems, and mediation between negotiation agents in a deadlock.
- *Participation agents* are responsible for facilitating communication between agents, agent negotiation and a KQML[3] interface for users to directly communicate with agents.
- *User agents* are agents interacting with the user, that should be developed by the user and must conform to the DIAS agent developer API.

DD2 **AMP specifications:**
The AMP should provide services for agents to interact and exchange messages and services, and register itself in other AMPs according to the agent place properties defined in the MASIF standard. Further, an AMP should initiate the required system agents needed to receive and register agents, facilitate inter-agent communication, monitoring agent activities, removing agents, and facilitate agent negotiation and mediation.

DD3 **Agent interface specifications:**
The CAGIS DIAS should provide an interface to other mobile agent systems defined in the MASIF standard. Further, an agent user client should be provided where the user can subscribe to AMPs, and initiate, configure, interact with and terminate her/his agents. An additional user interface for AMPs should also be provided for removing agents.

The CAGIS SimpleProcess Design Specifications

The CAGIS SimpleProcess is a workflow system used to model and enact individual workflow processes. Here is a summary of the full list of the design specifications (D44-D64) described in Appendix B:

[3]KQML is short for Knowledge Query and Manipulation Language

DSP1 **Architecture specifications:**
A process model in CAGIS SimpleProcess can consist of several distributed process fragments that can be locally modified and moved between workspaces.

DSP2 **PML specifications:**
The PML in CAGIS SimpleProcess should describe a process as a collection of process fragments consisting of one or more activities that can have the states waiting, ready and finished. Activities should execute code in HTML and are arranged in execution sequence through hyper-links.

DSP3 **Tool specifications:**
The CAGIS SimpleProcess should provide three tools: A *process server* that should manage activity states, adding/removing activities and facilitating movement of activities between workspaces; a *process modeller* that should enable the user to enter process models interactively; an *agenda manager* that should provide an agenda of activities and activate ready activities; and a *monitor tool* that should provide an interface for monitor process state and progress.

The CAGIS GlueServer Design Specifications

The GlueServer is a middleware used to facilitate interaction between software agents (CAGIS DIAS) and process fragments (CAGIS SimpleProcess). Here is a summary of the full list of the design specifications (D65-D75) described in Appendix C:

DGS1 **GlueServer specifications:**
The GlueServer should provide an interface enabling interaction with mobile agent systems, and an interface for providing interaction with workflow systems (CAGIS SimpleProcess). In addition, the GlueServer should be able to parse a GlueModel and react according to it.

DGS2 **GlueModel specifications:**
The GlueModel should specify relationships between an agent and a process fragment, the expected results returned from the agent, and what reactions to be executed based on these results.

4.4 Implementation

The implementation of the CAGIS PCE prototype has been carried out by the author of this thesis, and by several last-year students at the Department of Computer and Information Science at the Norwegian University of Science and Technology. In total, the CAGIS PCE prototype consists of almost 20,000 lines of code (where 20% of the code was written by the author and 80% by MsC students) in Java and Perl.

Most of the CAGIS PCE has been implemented in Java, making it possible to run the prototype on various platforms and making use of available technology based on Java.

In addition, we have used CORBA to integrate different parts of the CAGIS PCE, and XML for storing and representing information. The next three sections (sections 4.4.1 - 4.4.3) describes more in detail how each of the three main parts in the CAGIS PCE was implemented, and they have fulfilled the design specifications. The last section (section 4.4.4) gives a summary of how the CAGIS PCE was implemented and what parts of the CAGIS PCE provide support to the high-level requirements described in section 4.2.2.

4.4.1 Implementation of the CAGIS DIAS

A technology study briefly presented in CAGIS DIAS architecture section 4.3.3 suggested that the following technologies should be used in the agent architecture: Java and Java IDL for the component infrastructure, KQML and JATLite to provide inter-agent communication and AMP/workspace support, the Aglets framework to support mobile agents, and XML for information representation.

Our first implementation called DIAS I [PHBN99] tried to integrate these technologies into one working agent system. However, we discovered that it was impossible to directly integrate JATLite and Aglets because of overlapping functionality and that JATLite's functionality could not be used in a mobile fashion. Our decision was only to use the KQML layer in JATLite to provide KQML support and use the functionality provided in Aglets to provide AMP and workspace support. This was possible since JATLite is an open source system. In our first prototype of CAGIS DIAS, we managed to integrate Java, Java IDL, Aglets, the KQML-layer in JATLite and XML, but the services provided by the system were too low-level and simple. To create agents, the developers had to heavily use Aglets with some limited addition DIAS support for AMPs.

The second version of the agent prototype system was called DIAS II [HN00]. In DIAS II we improved the agent-API by providing a set of high-level methods, and ' by making the agent-API independent of the underlying technology (Aglets). We also added support for interactions with other mobile agent systems through the MASIF standard implemented in Java using ORBIX CORBA. The new high-level agent-API made it easier and faster to create new agents where the CAGIS DIAS system took care of localising agents (by using unique ID's or/and ontologies), communicating between agents, connecting and registering agents, etc. The new CORBA interface made it also possible to interact with the CAGIS GlueServer.

In 1999, IBM stopped supporting further development of Aglets making it impossible to run Aglets on newer versions of Java. To continue the development of CAGIS DIAS, we started to evaluate new technologies for providing mobility support for agents. The criteria we used for selecting a new technology to replace Aglets were (based on implementation experience from DIAS I):

- The technology should provide a high-level API, making it possible to implement an agent system faster.
- The technology should provide an easy transition from Aglets.

- The technology should be well integrated with Java.
- The technology should be easy to install and configure.
- The technology should handle dynamic changes of software system.
- The technology should be free and widely available.

Based on these criteria above, we chose to use JavaSpaces from Sun [FHA99] to implement DIAS III [SW00, Waa00, Sal01]. JavaSpaces is based on JINI, and is a framework for dynamic communication, coordination, and sharing of objects between network resources (more on JavaSpaces in section 2.7.1). JavaSpaces enabled us to replace Aglets quickly, and saved us to implement functionality for discovering agent clients and servers dynamically through the look-up service in JINI. Not all features in DIAS II have been implemented in DIAS III. For all implementations of DIAS, XML has been used to store agent information.

Testing the DIAS prototype

This paragraph describes how the prototypes CAGIS DIAS II and III were tested against the non-functional requirements and design specifications. We have not only used the CAGIS DIAS III when running scenarios because the implementation of this version of the prototype is not complete. The DIAS prototype has been tested against the design specifications given in Appendix A as well as the software development and maintenance scenarios described in paper presented in chapter 14, and the conference organising scenario described in the report presented in chapter 20.

Table 4.1 shows the results in percentage of completeness from testing the non-functional requirements presented in section 4.2.1 for the prototypes CAGIS DIAS II and III. The requirement N2 was not completely fulfilled in CAGIS DIAS II, because interface to other mobile agent systems was implemented using ORBIX CORBA which is not platform independent. N2 is fulfilled in CAGIS DIAS III, because this feature was not implemented. In addition, N5 is not completely fulfilled in CAGIS DIAS II because the Aglets framework demands manual configuration of the network-setup. This problem was solved in CAGIS DIAS III by using JavaSpaces and JINI.

Table 4.2 shows the results from testing the design specifications (summary of the design specifications are given in section 4.3.6, and the complete list is given in Appendix A) for the prototypes CAGIS DIAS II and DIAS III. The percentage is computed by the formula:

$$\text{Score} = \frac{\text{Number of implemented specifications}}{\text{Total number of specifications}} \circ 100\%$$

From table 4.2 we can see that most design specifications are implemented in CAGIS DIAS II, but support for mediation between negotiating agents is missing. However, the implementation of CAGIS DIAS III suffers from not being complete and is lacking of functionality for several AMPs and the implementation of the MASIF interface to other mobile agent systems.

Non-functional Requirements	**DIAS II**	**DIAS III**
N1 Openness	100%	100%
N2 Software and Hardware Requirements	80%	100%
N3 Response Time	100%	100%
N4 Security	100%	100%
N5 Maintainability and Extensibility	80%	100%

Table 4.1: The results from testing non-functional requirements in CAGIS DIAS II and III

Design Specifications	**DIAS II**	**DIAS III**
DD1 Agent specifications	93%, not D13, and D23	78%, not D6, D14, D15, D17, D24, and D25
DD2 AMP specifications	92%, not D38	67%, not D28, D33, D39, and D40
DD3 Agent interface specifications	100%	67%, not D41

Table 4.2: The results from testing CAGIS DIAS II and III design specifications

4.4.2 Implementation of the CAGIS SimpleProcess

The CAGIS SimpleProcess workflow tool implemented entirely in the programming language Perl, and the Common Gateway Interface (CGI) was used to offer a user-interface provided by a web-browser via a web-server. XML has been used to store process models, and small efficient file-databases provided in Perl have been used to manage state information for the process engine.

Testing the CAGIS SimpleProcess prototype

All the non-functional requirements described in section 4.2.1 are fulfilled in the implementation of the CAGIS SimpleProcess prototype. The prototype server must run on a software operating system running Perl and Apache web-server, but most operating systems do. Our prototype is also easy to maintain and expand, by adding CGI-scripts implemented in Perl if more functionality is needed.

The CAGIS SimpleProcess prototype has implemented the design specifications described in section 4.3.6 based on the full list of specifications given in Appendix B as given in table 4.3:

Table 4.3 shows that the specifications D47 “Process fragments can be distributed on different sites”, and D60 “The process server should enable moving activities from one workspace to another, and from one process server to another” were not fully imple-

Design Specifications	**CAGIS SimpleProcess**
DSP1 Architecture specifications	75%, not D47
DSP2 PML specifications	100%
DSP3 Tool specifications	89%, not D60

Table 4.3: The results from testing the CAGIS SimpleProcess design specifications

mented. D47 is not supported, because the current implementation of CAGIS SimpleProcess server does not provide inter-site support. The specification D60 is partly implemented, because movement of process fragments between workspaces is supported, but not movement of process fragments between sites.

4.4.3 Implementation of the CAGIS GlueServer

The GlueServer was also implemented in the Java programming language, and the GlueModels are stored in XML. The agent system interface was implemented with ORBIX CORBA according to the MASIF standard, using an interface agent in DIAS. The only current workflow interface has been provided through CGI.

Testing the CAGIS GlueServer prototype

The implementation of the CAGIS GlueServer covers all the non-functional requirements described in section 4.2.1 but not N1 fully. N1 is not completely fulfilled because the CAGIS GlueServer is not platform independent demanding ORBIX CORBA to run.

The CAGIS GlueServer prototype has implemented the design specifications described in section 4.3.6 based on the full list of specifications given in Appendix C as given in table 4.4:

Design Specifications	**CAGIS GlueServer**
DGS1 GlueServer specifications	66%, not D66, D72, and D73
DGS2 GlueModel specifications	100%

Table 4.4: The results from testing the CAGIS GlueServer design specifications

Table 4.4 shows that three design specifications are not implemented in the prototype of the CAGIS GlueServer. The specifications D66 and D72 are not fully supported because we have not tested and implemented any specific interfaces to other mobile agent systems. The F73 specification is not supported since the general workflow interface has not been implemented either.

4.4.4 Implementation Summary

The implementation of the CAGIS PCE is not a 100% complete implementation, but it contains the functionality needed to test this prototype and to demonstrate our approach. Table 4.5 shows what parts of the CAGIS PCE are used to support the high-level requirements described in section 4.2.2. Local work support is provided by the CAGIS SimpleProcess workflow tool (HF1), using simple modelling concepts and formalisms (HF3) that can be defined and changed by the user herself/himself (HF6). Cooperative work (HF2) is mainly supported by the CAGIS DIAS, but the CAGIS GlueServer is also used to specify executable cooperative rules between workspaces. The CAGIS GlueServer provides a flexible infrastructure for handling dynamic changes and re-configuration of process support (HF5) defined and managed by management representatives (HF7). All parts of the CAGIS PCE provide distributed work support, but the the CAGIS SimpleProcess workflow server is limited to be run on one single site. Section 5.1 and the report presented in chapter 20, describe more in detail how these requirements have been fulfilled when the CAGIS PCE was used to model and support a conference management process.

High-level Requirements	**Supported by**
HF1 Local work support	CAGIS SimpleProcess
HF2 Cooperative work support	CAGIS DIAS + CAGIS GlueServer
HF3 Distributed work support	CAGIS PCE
HF4 Light-weight support	CAGIS SimpleProcess
HF5 Dynamic support	CAGIS PCE
HF6 Local control	CAGIS SimpleProcess
HF7 Managerial control	CAGIS GlueServer

Table 4.5: Mapping high-level requirements to the CAGIS PCE implementation

We have spent most time on implementing the CAGIS DIAS prototype that also provides most advanced functionality such as mobile cooperative agents. Since the CAGIS DIAS, the CAGIS SimpleProcess and the CAGIS GlueServer have been developed in parallel, it has been hard to fully integrate them into one environment. During the projects of developing the prototypes we have used evolutionary development where we have explored and refined functionality during development. There has not been enough time to consolidate the implementation into one environment, forcing us to do some manual configuration and adjustments when running all three prototypes as an environment. We have however managed to demonstrate the most important features of the CAGIS PCE prototype that it is efficient to model and support processes with individual and cooperative activities, and that it is efficient to model and support dynamic process changes. More on this in section 5.1 and the report presented in chapter 20.

4.5 Paper Abstracts

This chapter presents abstracts of all the papers contributing to this thesis. All the papers have been published on international conferences and workshops, and have been fully reviewed. The papers have been divided into the two sections *Background Papers*, and *Core Papers*. The background papers were inputs to the work that resulted in the core papers. Note that some of the papers in the Core Papers section overlap. These overlaps were necessary to give the context of each paper, and made the papers easier to read when published as stand-alone papers in a conference proceeding. The overlaps are mainly the description of the CAGIS Process Centred Environment or parts of this environment. In addition, the Core paper section includes a report of an evaluation of the thesis that will be published as a paper in the future.

4.5.1 Background Papers

This section presents five papers that are relevant to the thesis, forming a basis when working on the core issues outlined in section 4.5.2. These five papers cover relevant topics like software process, software engineering, software process evolution, configuration management, awareness, cooperative conflict handling, conducting a case-study, implementing a workflow tool, and XML.

Paper 1: Total Software Process Model Evolution in EPOS

This paper presents a case study of a Norwegian banking software house where the objective is to adopt a categorisation framework for managing evolution in software projects to identify project profiles and evolution patterns, and to suggest improvements to better support frequent evolutions. Based on an analysis of collected evolution data from an ongoing case study, we elaborate a QIP-inspired method and own techniques to evolve corresponding process models in our Process-centred Software Engineering Environment, called EPOS. The method describes also how to synthesise and reuse evolution experience from completed projects to improve planning and estimation in new similar projects. The collected data demonstrates that requirement changes which are detected in later development phases, are major causes for cost overruns in the studied organisation.
Relevance to thesis: The work with this paper gave a good introduction to process evolution and how to deal with such changes in a Process centred Software Environment Environment (PSEE).
Author(s): Minh N. Nguyen (main), Alf Inge Wang, and Reidar Conradi.
Published where: International Conference of Software Engineering 1997 (ICSE'97), Boston, USA, 21-23 May, 1997 [NWC97].
My contribution: This thesis' author has written on most part of this paper, but in particular described how the EPOS PSEE can implement process changes.
Where in thesis: Chapter 8, page 109.

Paper 2: Planning support to Software Process Evolution

The ability to handle changes is a characteristic feature of successful software projects. The problem addressed in this paper is what should be done in project planning and iterative replanning so that the project can react effectively to changes. Thus the work presents research results in software engineering, as well as transfer of methods in knowledge engineering to software engineering, applying the AI planning technique to software process modelling and software project management. Our method is based on *inter-project experience* and *evolution patterns*. We propose a new classification of software projects, identifying and characterising ten software process evolution patterns and link them to different project profile. Based on the evolution patterns, we discuss the planning support for process evolution and propose several methods that are new or significantly extend existing work, e.g. **cost estimation of process changes, evolution pattern analysis**, and **a coarse process model for the initial planning- and the iterative replanning process**. The preliminary results have shown that the study of evolution patterns, based on inter-project experience, can provide valuable guidance in software process understanding and improvement.
Relevance to thesis: The paper focus forms a basis for understanding the changes in software projects and how to deal with these changes trough iterative replanning.
Author(s): Reidar Conradi, Minh N. Nguyen, Alf Inge Wang, and Chunnian Liu (main).
Published where: International Journal of Software Engineering and Knowledge Engineering. Volume 10, Number 1, 2000, World Scientific Publishing Company[CNWL00]. Also published [CNWL98].
My contribution: This thesis' author contribution to this paper is mainly on how to deal with project changes in process model (task-networks), and execution of experiments in EPOS.
Where in thesis: Chapter 9, page 129.

Paper 3: Improving Cooperation Support in the EPOS CM System

This paper reports our experiences gained in designing, implementing, and experimenting with technologies for improved support for cooperative work in our configuration management (CM) system. The aim of the work has been to find a set of mechanisms supporting cooperation in a range of situations, from planning and scheduling long-lasting CM activities, to resolving access conflicts between users. Although our tools are tailored for our home-grown environment, the general approach should be applicable also to other CM systems or usage domains. The emphasis of this paper is on flexible mechanisms to solve access conflicts without enforcing only one way of working.
Relevance to thesis: This work was a good introduction to awareness and cooperative support between people, forming a basis for cooperating agents in the CAGIS PCE.
Author(s): Alf Inge Wang (main), Jens-Otto Larsen, Reidar Conradi, and Bjørn P. Munch.
Published where: 6th European Workshop in Software Process Technology, Weybridge, UK, September 16-18, 1998 [WLCM98b].
My contribution: This thesis' author was the main author of this paper, and was responsible for implementing and executing experiments with cooperative support to resolve

access conflicts.
Where in thesis: Chapter 10, page 147.

Paper 4: Teaching Software Process Improvement through a Case Study

This paper describes the main design choices of a software process improvement course. The course is organised around an industrial case study. In addition it is based on lectures and group exercises. The case study is centred around four research questions: Why is process improvement important [in your company]? Which processes does your company have? Which improvement initiatives does your company implement? Which relationships exist between software improvement and software quality? During the case study, the students come in contact with actors from the local software industry. We experienced problems with with relating quality and process issues, and that the insight was too superficial. We also had problems with student involvement. Finally we propose a new set of questions that are: Briefly describe a software system that is (or has been) important for your company. Which attributes do you use to describe it? Which are the processes around this system? Which are the improvement initiatives around these processes? How general is that specific software system (and respectively its processes and improvement initiatives) in the context of your company?
Relevance to thesis: This paper was a good introduction to how to conduct a case-study. Also it was interesting to learn more about industrial software development processes.
Author(s): Torgeir Dingsøyr, Letizia M. Jaccheri, and Alf Inge Wang.
Published where: International Journal: Computer Applications in Engineering Education. Also published at International Conference on Engineering and Computer Education 99 (ICECE'99), Rio de Janeiro, Brazil, August 11-14, 1999 [DLW00].
My contribution: The author of this thesis has written on all parts of this paper, but was responsible for the sections *Introduction* and *Evaluation*.
Where in thesis: Chapter 11, page 163.

Paper 5: Using XML to implement a workflow tool

This paper presents experiences we had from building a workflow tool from scratch using XML technology. We will present some strengths found using XML-technology, but also some weaknesses. Although we had to create a simple process modelling language for this workflow tool, the focus of this paper is on experiences on using XML technology to build workflow tools. The experiences we have achieved, should be applicable for all kinds for process modelling languages. The paper consists of three main parts. First, the requirements for the workflow tool is outlined. Then XML technology is explained with some simple examples. The last part of the paper describes experiences we achieved from the experiment and the conclusions we drew from this.
Relevance to thesis: This work gave useful experiences and input to the workflow tool we implemented to be a part of the CAGIS PCE.
Author(s): Alf Inge Wang.
Published where: 3rd Annual IASTED International Conference on Software Engineer-

ing and Applications (SEA'99), Scottsdale, Arizona, USA, October 6-8, 1999 [Wan99]. There is also a reference to this article at Dr.Dobb's web-site, section for XML[4].
My contribution: Solo-paper.
Where in thesis: Chapter 12, page 173.

4.5.2 Core Papers

This section presents the papers that are directly related to the work on the CAGIS PCE consisting of a workflow tool, a multi-agent architecture, and a middleware for combining the two (GlueServer). As an introduction, *paper 6* describes how the CAGIS environment can be applied to a part of a conference organising process. This work was carried out together with two other members of the CAGIS project, namely Heri Ramampiaro and Terje Brasethvik. The CAGIS PCE is only *one part* of the whole CAGIS environment, which also consists of a document management system [Ter99], and a transaction management system [Rr99]. The *papers 7,8, and 9* describe the CAGIS multi-agent architecture for cooperative software engineering. These papers describe the architecture, the design, and some experiences from implementing this multi-agent architecture. *Paper 10* describes the CAGIS workflow system that supports distributed, mobile processes. The *papers 11 and 12* describe how the CAGIS multi-agent architecture and the CAGIS workflow system can be combined. This section also includes *report 1* that describes how our CAGIS PCE was evaluated together with two other PCEs applied on a conference organising scenario. This report will be published as a paper in the future.

Paper 6: Supporting Distributed Cooperative Work in CAGIS

This paper describes how the CAGIS environment can be used to manage work-processes, cooperative processes, and how to share and control information in a distributed, heterogeneous environment. We have used a conference organising process as a scenario and applied our CAGIS environment on this process. The CAGIS environment consists of three main parts: a document management system, a process centred environment, and a transaction management system. The paper describes how these main parts may be configured and used together in order to support cooperative work in distributed environments.
Author(s): Heri Ramampiaro, Alf Inge Wang, and Terje Brasethvik.
Published where: 4th IASTED International Conference on Software Engineering and Applications (SEA'2000), Las Vegas, Nevada, USA, 6-9 November 2000 [RBW00].
My contribution: The description of the CAGIS PCE, the description of the scenario and how the CAGIS environment is applied to the scenario.
Where in thesis: Chapter 13, page 187.

[4]Dr. Dobb's web-site section for XML: http://www.ddj.com/topics/xml

Paper 7: A Multi-Agent Architecture for Cooperative Software Engineering

This paper looks at how Cooperative Software Engineering (CSE) can be supported. We first investigate the process aspects by presenting a traditional process architecture supporting CSE. Then we propose a multi-agent architecture for CSE, which is better in terms of simplicity and flexibility, and particularly useful in modelling and providing support to cooperative activities. We describe an industrial scenario of CSE, and show how to apply the proposed architecture to this scenario. The scenario is based on a software development and maintenance process for a Norwegian software company.
Author(s): Alf Inge Wang, Reidar Conradi, and Chunnian Liu.
Published where: 11th International Conference on Software Engineering and Knowledge Engineering (SEKE'99), Kaiserslautern, Germany, 17-19 June, 1999 [WLC99].
My contribution: Main author.
Where in thesis: Chapter 14, page 203.

Paper 8: Design Principles for a Mobile, Multi-Agent Architecture for Cooperative Software Engineering

The paper describes experiences we have achieved from implementing a mobile multi-agent system for cooperative software engineering, based on the Aglets technology from IBM. When implementing the mobile multi-agent system, we faced problems dealing with locating agents, inter-agent communication, registration of agents etc. Based on our experiences, we present some design principles for how to locate agents, how agents should communicate, how to manage connection to the agent system, how to register agents and agent places, how to move agents, how to remove agents, and how to give CORBA-agent interaction support. These design principles should be applicable for others wanting to design mobile multi-agent systems using the Aglets technology.
Author(s): Alf Inge Wang, Anders Aas Hanssen, and Bård Smidsrød Nymoen.
Published where: 4th IASTED International Conference on Software Engineering and Applications (SEA'2000), Las Vegas, Nevada, USA, 6-9 November, 2000 [Alf00].
My contribution: Main author.
Where in thesis: Chapter 15, page 217.

Paper 9: Implementing a Multi-Agent Architecture for Cooperative Software Engineering

The paper describes experiences we have earned from implementing a multi-agent architecture used to support cooperative software engineering. Before starting to implement a multi-agent architecture, important decisions and considerations must be taken into account. Decisions on how to provide efficient inter-agent communication support, what language should the agents talk, should the agents be stationary or mobile, and what technology should be used to build the architecture must be made. This paper describes how we implemented our multi-agent system, and the experiences we gained from building it.
Author(s): Alf Inge Wang.
Published where: 12th International Conference on Software Engineering and Knowl-

edge Engineering (SEKE'2000), Chicago, USA, July 6-8, 2000 [Wan00a].
My contribution: Solo-paper.
Where in thesis: Chapter 16, page 229.

Paper 10: Support for Mobile Software Processes in CAGIS

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.
Author(s): Alf Inge Wang.
Published where: 7th European Workshop on Software Process Technology (EWSPT'2000), Kaprun, Austria, February 22-25, 2000 [Wan00b].
My contribution: Solo-paper.
Where in thesis: Chapter 17, page 243.

Paper 11: Integrating Software Process Fragments with Interacting Agents

Cooperative software engineering processes involve structured, repeatable processes as well as dynamic, cooperative processes. Existing workflow systems are suited to model and support the former type of processes, and multi-agent systems are suited to model and support the latter. We have designed and implemented a gluing-framework for integrating workflow processes with software agents. By using this framework, support for cooperative software engineering processes can be provided in a better and more flexible way. This paper focuses on how to integrate these two kinds of components into a functioning multi-agent based cooperative software engineering system.
Author(s): Alf Inge Wang, Reidar Conradi, and Chunnian Liu.
Published where: 4th IASTED International Conference on Software Engineering and Applications (SEA'2000), Las Vegas, Nevada, USA, 6-9 November, 2000 [WCL00].
My contribution: Main author.
Where in thesis: Chapter 18, page 259.

Paper 12: Using Software Agents to Support Evolution of Distributed Workflow Models

This paper outlines a high-level design of how software agents can be used combined with an existing CAGIS Process Centred Environment to deal with evolution of distributed, fragmented workflow models. Our process centred environment allows process fragments of the same workflow model to be located in workspaces that are geographically distributed. These process fragments can be changed independently in local workspaces causing consistency problems. We propose to use software mobile agents, offering aware-

ness services solving conflicting updates of process fragment. Our solution is illustrated using some scenarios.
Author(s): Alf Inge Wang.
Published where: International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000) at International ICSC Congress on Intelligent Systems and Applications (ISA'2000), Wollongong (near Sydney), Australia, December 12-15, 2000 [Wan00c].
My contribution: Solo-paper.
Where in thesis: Chapter 19, page 273.

Report 1: Evaluation of a Cooperative Process Support Environment

This report describes an evaluation where the same distributed conference organising process is modelled in three different process centred environment Endeavors, ProcessWeb, and our own CAGIS Process Centred Environment. Endeavors is an activity based, flexible workflow system, ProcessWeb is a role-based workflow system with a web-interface, whereas the CAGIS Process Centred Environment combines an activity based workflow system with a software agent system. The goal of the experiment is to investigate if a combination of a traditional workflow system and software agent system better can model and support distributed cooperative processes than stand-alone workflow systems. We also want to investigate if a combination of workflow system and agent system better can adapt to occurring process changes. A conference organising scenario is used as a case in the experiment because it illustrates a distributed process containing both simple, individual activities as well as more dynamic, cooperative activities. By evaluating how well the different process centred environments can model and support the scenario, as well deal with process changes, our CAGIS Process Centred Environment is validated.
Author(s): Alf Inge Wang.
Published where: Technical report IDI-nr 9/00, Dept. of Computer and Information Science, Norwegian University of Science and Technology, December 2000.
My contribution: Solo-report.
Where in thesis: Chapter 20, page 285.

CHAPTER 5

Thesis Evaluation

This thesis has presented an approach for improving cooperative process support for distributed users in a heterogeneous environment. Our approach combines traditional activity-based workflow with software agents. By doing so, we can efficiently model and enact individual activities as well as cooperative activities (more on this in section 5.1). We have implemented a middleware called GlueServer to enable flexible configuration of interaction between workflow systems and agent systems. The GlueServer enables us to integrate dynamic agents as a part of the workflow. These agents can be used for typical cooperative tasks (e.g. negotiation of resources and coordination of artifacts), as well as environment observers notifying the workflow system for occurring events.

To evaluate our CAGIS PCE, we have modelled a conference organising scenario in our process environment as well as in two others. In addition, we have run these process models in all three process environments to see that the models are enactable and that they can provide user support. We cannot regard the execution of the process models as real enactment because they are not run in their real environment for actual usage. Within the CSCW and workflow community there have been conducted some similar experiments, but within SPT research we have not been able to find any similar. We hope that this experiment can be an encouragement for others to conduct real process enactment experiments.

This chapter is organised as the following: Section 5.1 describes our evaluation of the CAGIS PCE by comparing it to two other systems, section 5.2 describes how our CAGIS PCE addresses the research questions that were identified in the introduction of this thesis, and finally section 5.3 summarises the contribution of this thesis.

5.1 Comparing the CAGIS PCE with two other PCEs

In the report "*Evaluation of a Cooperative Process Support Environment*" (see section 20), we compared our approach (the CAGIS PCE) to two workflow systems, Endeavors and ProcessWeb, by modelling the same conference organising scenario. The process models in the Endeavors are activity based and are created by using a graphical modelling tool drawing an activity network and specifying attributes of the activities. When an activity in Endeavors is enacted, an activity handler is executed. The activity handlers can be a wrapping of a commercial tool (e.g. a spreadsheet) or for instance a Java program with a graphical user interface (e.g. a form). The activity handlers interact with Endeavors through events (messages). The PML in ProcessWeb is a special object-oriented programming language where the process is represented as roles and interactions between roles. A role is defined by some actions (methods), resources (attributes), and guards (preconditions) that are used to specify when actions should be executed. Interactions provide communication channels between two roles.

The conference organising scenario was described as a process consisting of individual activities (such as "Make call for papers", "Handle received paper", "Fill in review report" etc.) and cooperative activities (such as "Reviewer allocation", "Session allocation" etc.). The purpose of the evaluation was to answer to research questions: Is a combination of a traditional workflow system and a software agent system better compared to a stand-alone workflow system to:

R1 Model and demonstrate enactment of processes containing both dynamic cooperative activities, as well as structured, individual activities ?

R2 Adapt to process changes ?

The evaluation of the two research questions listed above was answered by using a combination of quantitative and qualitative research method. By *quantitative* is meant that research question R1 was measured by looking at how much of (coverage) the conference organising process could be modelled and enacted by each PCE (measured in percentage), and the time spent on modelling the process. To find the answer to research question R2, adaptability was measured by assessing the effort spent on implementing a specific process change using a scale 1-5, where 5 indicated very little effort and 1 indicated very strong effort. As the statistical data for this experiment were insufficient because only one scenario was modelled, *qualitative* discussions were also used to to evaluate the research questions when comparing the results from the experiment. The empirical data were used to give an indication of the differences, and reasons for these differences were given when evaluating the experiment.

5.1.1 Coverage of the Scenario

Table 5.1 shows results from measuring coverage and modelling time for performing the conference scenario in the three PCEs. The modelling time is not objective since we did most of the modelling ourselves. In research question R1, we wanted to evaluate how complete the three different PCEs could model and support the conference organising scenario. All PCEs were able to completely model all individual activities, while we experienced some problems modelling cooperative activities for allocating papers and timeslots (resource negotiation) in Endeavors. In the conference management scenario, reviewers can pick what papers they want to review, and the paper allocation activity initiate negotiations between reviewers if more than three reviewers have selected the same paper. Our problem was how to represent a negotiation process between different roles in an activity-network. We chose to solve this problem by sequentially checking the need for a negotiation represented by an allocation activity for each role. After checking all roles (PC Member 1 to 5), we checked if any remaining negotiations were required. If more conflicts remained, the same allocation activities were executed once again. This approach is illustrated in figure 5.1.

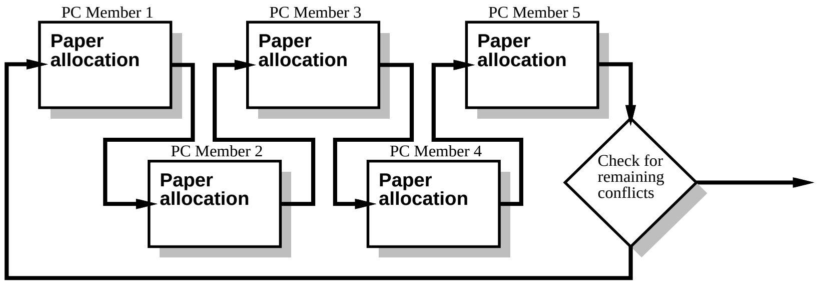


Figure 5.1: Cooperative activities modelled in Endeavors

Modelling	**Endeavors**	**ProcessWeb**	**CAGIS PCE**
Individual activities	100 %	100 %	100 %
Cooperative activities	80 %	100 %	100 %
Modelling time for individual activities	6 hours	10 hours	3 hours
Modelling time for cooperative activities	10 days	2 days	5 days

Table 5.1: Coverage of the scenario modelled

The modelling time for individual activities in table 5.1 shows that most time was spent in ProcessWeb, then Endeavors, and least time was spent in CAGIS PCE. An explanation could be that the modeller was most familiar with CAGIS PCE, but the modelling in Endeavors and ProcessWeb was very straight forward without any time-consuming problems. The difference in modelling time can be explained as following:

- **ProcessWeb:** The PML in ProcessWeb is close to a textual object-oriented programming language, meaning that the modeller needs to "program" the process as roles with different states representing the activities that the role is responsible for. In addition, the modeller had to implement the infrastructure for interaction between roles, role assignments, and role configurations.

- **Endeavors:** More time was spent on modelling in Endeavors compared to CAGIS PCE, because in Endeavors we had to implement an activity handler that provided a user-interface (a wrapping of a Web-browser) for guiding the user. If we don't consider the time spent on implementing the activity handler, Endeavors was more efficient for modelling the individual activities than CAGIS PCE. This means that if you have the required activity handlers before you start to model processes, Endeavors is the most efficient for modelling individual activities.

According to table 5.1, ProcessWeb was the most efficient PCE for modelling cooperative activities. The reason for this is that the PML used in ProcessWeb is designed to efficiently implement role-interaction. CAGIS PCE uses an agent-API to implement the cooperative activities which in current state, are too low-level for modelling cooperative activities efficiently. In Endeavors, the cooperative activities were implemented in Java from scratch, making this the least efficient approach. The developers of Endeavors have suggested to use globally visible blackboard for sharing and negotiating about common objects accessed by various roles. This was not implemented because the documentation for implementing such blackboards was not available.

5.1.2 Adaptability of Process Change

This subsection describes how efficient the three PCEs can adapt to a set of process changes described in section 20.5.11 in the report presented in chapter 20. All PCEs have support for changing the process model during enactment. Table 5.2 describes the results from evaluating three PCEs' ability to process changes. The result reflects how much effort must be spent to implement the change, and is judged on a scale 1-5 (where 5 is *Very little effort* and 1 is *Very strong effort*):

Process change	**Endeavors**	**ProcessWeb**	**CAGIS PCE**
1. Change activity sequence	5	3	5
2. Assign activity to another role	5	2	5
3. Change negotiation strategy	1	3	5
4. Change reviewer selection	1	3	5

Table 5.2: PCE adaptability to Process changes

Here are comments on results for each process change (1-4) shown in table 5.2:

1. **Change the activity sequence:** This process change in **Endeavors** is done by a graphical manipulation of the activity-network. In **ProcessWeb** this particular process change depends on how the activities are represented in the roles. If the activities are represented as states in the role, some if-sentences must be changed to alter the activity sequence. The changed role definition must in addition be compiled into the system, and the role must be modified to get the new behaviour. In **CAGIS PCE**, to change an activity sequence, you simply state where you want to move the activities (the process fragment).

2. **Assign an activity to another role:** In **Endeavors** role assignment can be changed by editing the *AssignedTo* attribute for an activity through Endeavors' graphical user interface. In **ProcessWeb** assigning an activity to another role is more complicated. First, the PML code describing the activity must be transfered from one role to another. Second, the PML code for both roles (source and target roles) must be changed to cope with the removal of activity in the source role and adding the activity in the target role. Third, PML code for both roles must be compiled, and both roles must be modified to their new behaviour. The reason why it is hard to assign an activity to another role in ProcessWeb, is that the role is the unit of change. This problem can be avoided by modelling all activities as separate roles. In **CAGIS PCE**, hierarchical workspaces are used to represent roles. Re-assigning an activity to another role, is done by simply moving the activity to another workspace.

3. **Change negotiation strategy:** We have not considered the effort of implementing the new negotiation strategy, but rather how to integrate an already implemented negotiation strategy into the process. In **Endeavors**, it is rather hard to implement the N2 negotiation strategy (described in section 20.5.11). This is because it is hard to represent a network of interacting roles using an activity-based process formalism. The new negotiation strategy demands communication across different roles, and is hard to represent in an activity network. **ProcessWeb** is probably the best environment for implementing the new negotiation strategy, but to incorporate this strategy with existing roles can be hard. N2 demands extensive changes in the roles that can make it hard to migrate from an old role definition to new one. If this process change was known in advance, the process model could be implemented to handle such massive changes. It should be noted that this problem has been addressed in ProcessWeb through support for meta-process. A process architecture is used to provide support for a generic change making migration easier. Essentially, making one change is hard, but with a meta-process you can spread this cost over many changes over the lifetime of your system. In **CAGIS PCE**, it is very easy to change negotiation strategy by editing the GlueModel for the process fragments involving negotiation agents. To implement the N2 in the agent system is probably more time consuming than using ProcessWeb, but it is really easy to change negotiation strategy if a matching implementation (cooperative pattern) is already available.

4. **Change reviewer selection:** This particular process change demands that an activity is monitored and if the allocation of papers (negotiation) among reviewers is not

complete within a certain time, the PC Chair will select the reviewers for each paper. In **Endeavors** this process change can be implemented by adding timeout functionality for the activity handlers in the paper allocation activity. When the timeout event occurs, a control activity can be used to route the workflow to PC Chair that have to manage the reviewer selection on her/his own. This process change demands a lot of changes in both the activity handlers and the activity network. In **ProcessWeb** this process change is easier to implement, but will also demand some effort. The role assigned originally to this activity (PC Member) needs to have a timeout function for selecting and allocating reviewers to papers. When this timeout function is triggered, a message is sent to the PC Chair role that she/he must allocate reviewers to papers. This means that the PC Chair and the PC Member role definitions must be changed and the communication channel between them must be configured. In **CAGIS PCE** the GlueModel can be used to implement this process change. A monitor agent will monitor the reviewer allocation agents, and if they are not finished within a certain time, the monitor agent will notify the GlueServer. The GlueServer will then initiate an execution of a reviewer selection process fragment for PC Chair. The definition of the reviewer selection process fragment for PC Chair must be defined. Since monitor agents are a part of the CAGIS DIAS, changes must be made in the GlueModel, and an additional process fragment for reviewer selection for PC Chair must be defined.

5.1.3 Reflections on the Evaluation

Looking back at the research questions in section 1.3, we can say from the results in the two above sections that a combination of a traditional workflow system and a software agent system is better compared to a stand-alone workflow system to model and enact cooperative processes and adapt to process changes. Some objections can, however, be raised to this statement:

O1 **The selection of scenario is biased.** The reason for selecting the conference organising scenario was to use a scenario that was already described in the literature (external validity). One important question is to ask whether the conclusion of this evaluation is valid for other scenarios describing cooperative processes. To answer this question, we have to look at how the conference scenario has been modelled. We have distinguished between individual and cooperative activities. Individual activities are activities where one role is assigned for performing this activity. In cooperative activities, several roles are involved in performing the activity. We believe that as long as other scenarios are modelled by distinguishing between individual and cooperate activities, the evaluation result will be valid. If another approach is used to model the process, the evaluation result is not necessarily valid.

O2 **More experience with your own environment.** A problem with our experiment, is that we are more experienced with our own environment than the others. This means that the modelling time can be unreliable. To address this problem, we spent about one week to exercise process modelling in Endeavors and about one month with modelling process in ProcessWeb. The reason why more time was spent on ProcessWeb, was that the author had used ProcessWeb for modelling before the experiment was planned. In addition, we had no time-consuming problems when modelling the individual activities in Endeavors and ProcessWeb. For cooperative activities a modelling expert in ProcessWeb was used, but not for Endeavors. This means that the modelling time for cooperative activities in Endeavors is likely to be less if an expert had been used. However, since modelling interaction between roles in Endeavors is not a part of the process modelling language, it is likely that the evaluation result will be the same.

O3 **Statistical invalid data.** Using only one target process (conference process) we can not statistically validate our two research questions only based on the quantitative data we have collected. For a statistically valid experiment, data should be collected from several scenarios modelled in the three PCEs. The collected data have only been used as an indication of the evaluation result, and qualitative discussions have enlightened the results from data collection.

O4 **The selection of process changes is biased.** We have tried to pick out process changes that are likely to occur in the conference scenario. It is possible that our selection is too limited, and more changes should be considered. Other process changes could have indicated a more nuanced score for the three PCEs. However, we believe that the selected process changes represent a wide spectrum of possible changes making the result believable.

It should be noted that Endeavors and ProcessWeb are much more mature environments compared to CAGIS PCE, and they offer richer semantics for expressing the process. In CAGIS PCE, we have used agents to provide the functionality that we lose through having only a simple workflow model. Since the agent-API is still too low-level, more advanced workflow can be time consuming to implement in CAGIS PCE.

5.1.4 Comparison Conclusion

In this section we have endeavoured to investigate if a combined workflow - agent system is more able to model cooperative processes and to implement process changes compared to a stand-alone workflow system. We have modelled a conference organising scenario in the PCEs Endeavors, ProcessWeb, and CAGIS PCE, and collected some data during the modelling of the scenario. Our collected data and discussions indicate that the combined approach performs very well in respect to cooperative processes and process changes. This result is an encouragement to continue our research.

This evaluation was mainly performed by the author. For similar future evaluations, we would like to pick people to do the experiments without any prior knowledge to either system to ensure the validity of the results. One approach could be to use students that are not familiar with any of the PCEs, give them a proper introduction to the PCEs (the same amount), and let them model some processes. By using this approach, we would be able to collect more data, and could make some statistical analysis. A problem would be to find and pick descriptions of neutral scenarios that do not favour one of the PCEs. Another approach would be to use experts on each PCE to model processes. A problem with this approach is to deal with interpretations of the scenarios. Also if the experiment was not controlled, it could be possible to fake the measurements. We can conclude that doing such experiments are really hard and time-consuming.

The main contribution of our work is the GlueServer, making it possible to define couplings between activity-based workflow and software agents. With regards to efficiency, ProcessWeb was the best environment to model cooperative activities. For activities where activity handlers were already implemented, Endeavors was most efficient for modelling individual activities. Future work should therefore investigate how a combination of Endeavors, ProcessWeb and CAGIS GlueServer would work (federation of workflow systems). This combination features solid and rich modelling support for both individual and cooperative activities, where the CAGIS GlueServer acts as a middleware. Further, the GlueServer can be used to combine more than two workflow systems, allowing loosely coupled, autonomous groups to choose their own workflow tool. In this way, the CAGIS PCE provides cooperative support for a heterogeneous environment. Current implementation for the CAGIS GlueServer does not yet provide a federation of workflow systems through Workflow Management Coalition's interoperability workflow-XML binding. Other agent systems can also be integrated through the implementation of OMG's MASIF interface.

5.2 Propositions to our Research Questions

In section 1.3 ten research topics were outlined, describing the focus of the research in this thesis. Here is a summary of how these research questions have been addressed:

1. **Modelling: Investigate what is needed to model and enact distributed cooperative processes.** We have suggested to represent the process by software agents, by an XML-based workflow model and a GlueModel. The decomposition of this research question is addressed as following:

    - **1a) Formalism needed to model and enact individual processes**. We model individual processes as a network of related activities written in XML. Our XML-based process modelling language is described in paper 10 located in chapter 17.
    - **1b) Formalism to model and enact distributed, cooperative processes**. We have proposed to use software agents to model and support cooperative processes. In addition we use a GlueModel to define the interaction between individual processes and cooperative processes. Paper 7 in chapter 14 describes our cooperative agent approach, and the GlueModel is described in paper 11 in chapter 18.

- **1c) Process distribution among participants**. The process can be distributed in workspaces. Since every activity has an unique identifier (URL / Workspace / Activity), the process for individual participants can be distributed. Paper 10 in chapter 17 describes our approach for distributing processes.
- **1d) Model and support dynamic process changes**. Dynamic process changes are supported as following: 1) Software agents are used to probe the environment for specified events or changes that will be reported to the GlueServer, 2) The GlueServer will execute a reaction defined in the GlueModel, 3) A reaction can be used to re-configure the process model, change parts of the process model, execute specified parts of the process model, initiate another agent, change the GlueModel. The support for dynamic process changes is described in paper 7 (chapter 14), paper 10 (chapter 17), paper 11 (chapter 18), and paper 12 (chapter 19).

2. **Tools: Investigate how to create an infrastructure for providing distributed process support.** Our process support infrastructure consists of three main components: A workflow tool, a multi-agent system, and a GlueServer. The decomposition of this research questions is addressed as following:

   - **2a) Architectures needed to provide execution support for individual processes**. For the execution of individual processes, we use a CGI-based process server that interacts with the users through standard web-browsers. More details about this architecture can be found in paper 10 in chapter 17.
   - **2b) Architectures needed to provide execution support for distributed, cooperative processes**. We use a multi-agent architecture for this purpose. In addition, the GlueServer provides a middleware between the multi-agent architecture and the workflow tool. The CAGIS multi-agent architecture for cooperative engineering is described in paper 7 (chapter 14), paper 8 (chapter 15), and paper 9 (chapter 16).
   - **2c) Technology for providing a distributed, heterogeneous environment**. We have proposed to use the Web, Java-based agents, and CORBA to be able to cope with various platforms and systems. Technology issues for the CAGIS multi-agent architecture are discussed in paper 9 in chapter 16.
   - **2d) Combinations of technologies (2a,2b and 2c)**. Our CAGIS PCE is implemented by combining different technologies using middleware as CGI, Web, XML, and CORBA. An overall description of the CAGIS process centred environment is presented in paper 6 in chapter 13, and in paper 13 in chapter 20.
   - **2e) Open process architecture**. We suggest to use the WfC XML-interoperability standard, and MASIF to enable other workflow systems and other agent systems to interact with our environment. You can find more details on this in paper 10 in chapter 17, and paper 11 in chapter 18.

3. **Validation: Investigate how our approach compares with others**. We have modelled the same conference management scenario in three different process centred

environments, including our own. Further, we have evaluated these environments according to model completeness, and ability to handle process changes. The result showed that our approach consists of a useful combination of process support (for individual, cooperative activities, and for dynamic changes), but each part of our environment could be improved (CAGIS DIAS and CAGIS SimpleProcess) at their particular process support areas. The evaluation of our approach is presented in report 1 in chapter 20. In addition, we have also modelled other processes using our CAGIS PCE showing that it is not limited to only supporting conference management processes.

5.3 Summary of Contribution

Here is a list that summarises the contribution of this thesis:

- Seven papers describing our CAGIS PCE that are published at international conferences and workshops. In addition, five other papers related to the thesis have been published internationally, where two of them are journal papers.
- A multi-agent architecture and prototype implementation to support cooperative processes in software engineering and other domains.
- A prototype implementation of a flexible workflow tool for distributed processes.
- A simple XML-based PML.
- A gluing middleware framework and implementation for handling interaction between several single workflow systems and a multi-agent system.
- An XML-based language for specifying GlueModels (relations between process fragments and software agents).
- An application and evaluation of the framework, where the framework is applied to a conference organising process, a cooperative software engineering process, and a project organising scenario. Further, the framework was evaluated against two other process support systems, namely ProcessWeb and Endeavour. The result of this evaluation showed that our framework is useful for modelling and supporting cooperative processes and for handling process changes.

CHAPTER 6

Future Work

In the thesis evaluation in section 5, we said that our GlueServer facilitates several workflow systems and agent systems to co-exist in the same environment. This means that the GlueServer acts as a *federation server* for various process systems. We know that the GlueServer is technically capable of facilitating this federation of different systems, but we need to make larger experiments to see how well this integration works for real scenarios. Future work should explore scenarios where several different systems are involved in a heterogeneous environment, loosely coupled through the CAGIS GlueServer.

Our current implementation of the CAGIS DIAS (agent system) is rather immature for efficiently model cooperative activities. Thus, further work on the CAGIS DIAS should enhance the agent-API to make it easier to model and implement cooperative activities. Another approach is to use another existing mature cooperative agent platform, or simply to use a role-based workflow tool as ProcessWeb for the purpose.

Mobile devices like portable PCs, palm-top PCs, and mobile-phones are becoming more and more popular. An increasing number of applications will be available on such devices. This means that future process support environments should consider support for mobile devices. Mobile devices also mean that people's work can be carried out at different places, allowing people to work in virtual companies or virtual organisations. This means that the process modelling languages should take into account *mobile work* as a part of the formalism. Another issue that must be considered is how to provide process support for a variety of devices with different hardware, operating system, screen sizes, input devices etc. Also mobile devices are usually not on-line all the time, bringing up problems with synchronisation of models and files, update of process changes etc. A new Norwegian research project named Mobile WOrk Across Heterogeneous Systems (MOWAHS) starting June 2001 will bring the research from the CAGIS project into a mobile world. Industrial cooperation with the MOWAHS project is being solicited.

CHAPTER 7

Concluding Remarks

Since the sixties, the software industry has struggled with creating huge, reliable and maintainable software systems on time, and on budget. Also today, the software industry is struggling with the same problems, and no ultimate solution is found. In this thesis, the focus has been on how to model and support development processes, and how to deal with changes of these processes. Software development processes are highly creative and human-centric, and these factors should be taken into account by PCEs supporting software development. Thus, such PCEs must provide modelling languages and the execution platform to execute cooperative processes that can change when they are enacted. We believe that our approach presented in this thesis represents a framework in the right direction for better supporting software development processes. Further experimentation and exploration of our CAGIS PCE will give us indications whether this is the right way to go.

Part II

Background Papers

CHAPTER 8

Total Software Process Model Evolution in EPOS

Minh N. Nguyen, **Alf Inge Wang**, and **Reidar Conradi**[1]

Abstract
This paper presents a case study of a Norwegian banking software house where the objective is to adopt a categorization framework for managing evolution in software projects to identify project profiles and evolution patterns, and to suggest improvements to better support frequent evolutions. Based on an analysis of collected evolution data from an ongoing case study, we elaborate a QIP-inspired method and own techniques to evolve corresponding process models in our Process-centered Software Engineering Environment, called EPOS. The method describes also how to synthesize and reuse evolution experience from completed projects to improve planning and estimation in new similar projects. The collected data demonstrates that requirement changes which are detected in later development phases, are major causes for cost overruns in the studied organization.

Keywords:
Process model evolution, experience reuse and learning, categorization framework for process evolution, evolution pattern, empirical evolution data.

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7035 Trondheim, Norway. Phone: +47 73593444, Fax: + 47 73594466, Email nguyen/alfw/conradi@idi.ntnu.no

8.1 Introduction

The software industry needs to develop high quality software predictably on time and budget. Much research has therefore focused on technologies for Software Process Improvement (SPI), i.e. techniques for improving the productivity and quality of associated processes. We can mention efforts such as SEI-CMM [PWCC95], QIP/GQM [BCR94b], Bootstrap [HMK$^+$94] and SPICE [Dor93], etc. Likewise, many case studies have been conducted in software organizations to validate the applicability of proposed technologies [BCM$^+$92] [PC94]. However, there is insufficient research addressing the innumerable and unforeseen process changes that occur during normal software projects. Frequent changes and interruptions are considered as major cause for late delivery, cost overrun, missing features, and thus poor quality. The ability to handle unexpected events occurring both within the organization and in the surrounding environment is thus claimed to be a characteristic feature of successful companies. We need to improve our ability to predict, plan and manage changes based upon previous experiences. Adequate enactment support for process changes, embedded in a PSEE, is also considered necessary and desirable.

Our previous work [NC94a] has been revised to contain a classification of process change and their impact on the enacting process (project) in the form of recognizable evolution patterns. Such patterns associated with typical project and product profiles will contribute to building an empirical base. By utilizing this base the predictability in project planning/scheduling process is improved and more confident. We have collected empirical evolution data from several software projects to provide us better understanding of the actual evolution profile. Appropriate support has then been introduced in our PSEE to better manage the observed evolution.

8.2 Related Work

Some research has been dedicated to process model evolution. In [JC93] and [BFG93], basic mechanisms and techniques for process model evolution are identified and implemented in two different PSEEs, EPOS and SPADE respectively. These efforts have emphasized methods and mechanisms for changing process model fragments, represented as types or classes in a versioned repository. No concrete couplings have been made with actual evolution in real software projects. On the other hand, Madhavji provided a methodological perspective of evolution [Mad91] and environmental facilities (e.g., dependency and change structure) for changes in the Prism model [Mad92]. The model focuses on managing consistent change propagation in a feedback-based environment. Unfortunately, little effort has been dedicated to pursuit and apply the work which still remains at model level. The major focus in [LB85] and [Leh94] is on classification and studies of evolving entities, especially software systems, referred to as Program Evolution Dynamics. In this work, Lehman has identified five evolving entities in the software process: revision/version, S-type program[2], E-type application[3], process, and process model. However, this work does not offer sufficient detail in systematizing the impacts of evolution patterns in a concrete context. Work by DeMarco [DeM82], in the TAME project [OB92] and empirical studies at NASA-SEL [MPsP$^+$94] discuss how to achieve control and to improve estimation accuracy in rather large software projects. However, these results are not necessarily applicable for small and medium enterprises (SMEs). Large software projects are far more complex due to number of persons, number of software components and required amount of management involved. In general, there is little progress in obtaining and keeping control over evolving software processes by exploiting technologies from software measurement [Fen91] [Cap91] or from experience reuse/synthesis [BR91]. Indeed, most project management tools are of limited practical value, due to the accumulated effect of process changes during project execution. Thus, EPOS attempts to integrate process and project support in managing changes during development.

8.3 Conceptual and Categorization Framework

We distinguish between a changing *real world* where managerial and technical activities take place, and a *modeled world* ,where the human perceptions and constraints of the real world is represented by models and documentation. The former world is continuously evolving due to changing needs and perceptions. The latter model remains static, until humans determine to change it according to the world it reflects.

A *software process* is a set of managerial and technical activities applying certain technologies (methods and tools) to transform a requirement into a software system. The process consists of three parts: a *production process* (p-p), a *meta-process* (m-p), and process support (p-s). The primary goal of the p-p is to develop a software system in a project context with limited *resources* (humans, production tools) and *budget* (scheduled cost and time). Furthermore, the p-p should adhere to a *plan* acting as a project-specific process model. The m-p describes and governs overall *planning* and *executing* of p-p, as well as *packaging* of gained experiences and *evolving* the entire software process. The p-s is comprised of a model of the real process (p-p and m-p) expressed in process modeling language(s) and manipulated by a set of process tools (a PSEE).

There are two types of human agents interacting with and continuously raising change pressure to a software process. *External agents* consist of suppliers/sub-contractors and competitors. *Internal agents* comprise senior people (executive managers); middle (line and project managers); process (QA, process designer); SW engineers (analysts, designers, developers, testers, service people, etc.), and customer representatives. *External factors*, such as market trends/forces, technology availability, and unanticipated delays/distortions, also influence and drive process evolution.

Below, we present a **categorization framework** for process evolution. A detailed de-

[2]S-type program is a program which satisfies the fixed initial specifications.

[3]E-type application is subjected to continuous evolution as environmental conditions change.

scription can be found in [NC96]. Our framework distinguishes between *where, why, what, when, how* and *by-whom* process changes are introduced. Those six **dimensions** can be hierarchically decomposed for refinement and future extensions.

Where: identifies the sources that request or cause a given process change.

Why: represents the major causes (drivers) behind changes. They are used for causal analysis.

What: describes what process parts (p-p, p-s, m-p) are requested to be changed or affected by this action.

When: distinguishes between the time when the change request is detected (Change Detection Time - CDT) and the time when the proposed change is designed and implemented (Change Realization Time - CRT).

How: records the corrective and preventive actions being conducted to handle a given process change. It contains organizational change, technological innovation or plan adjustment.

By-whom: identifies the human agents who approve and perform the change action.

Each dimension can be further decomposed into several aspects. Each aspect is represented by appropriated categories. An observed process evolution, categorized by the proposed framework, is called an **evolution pattern**. The evolution pattern is instrumented by a cost measure in term of gain or loss of productivity or progress. Figure 8.1 depicts the elaborated categorization framework for process evolution which is used in the case study.

8.4 EPOS

This section gives you an overview of the PSEE EPOS and how EPOS support process evolution through mechnisms and tools.

8.4.1 EPOS System Overview

EPOS [MCJOLW95] [CHLN94] is a software process modeling and enactment system. EPOS supports a reflexive, object-oriented software process modeling language called SPELL [C^+92]. Several different sub-models are supported in EPOS for describing activities and products. These sub-models are: *Activity (task) model, Product model, Tool model, Human and Role model, Cooperation model* and *Meta-Process model*.

To support process modeling and evolution, we facilitate basic mechanisms for incremental (re)planning and enactment of the process models by process tools like the Planner

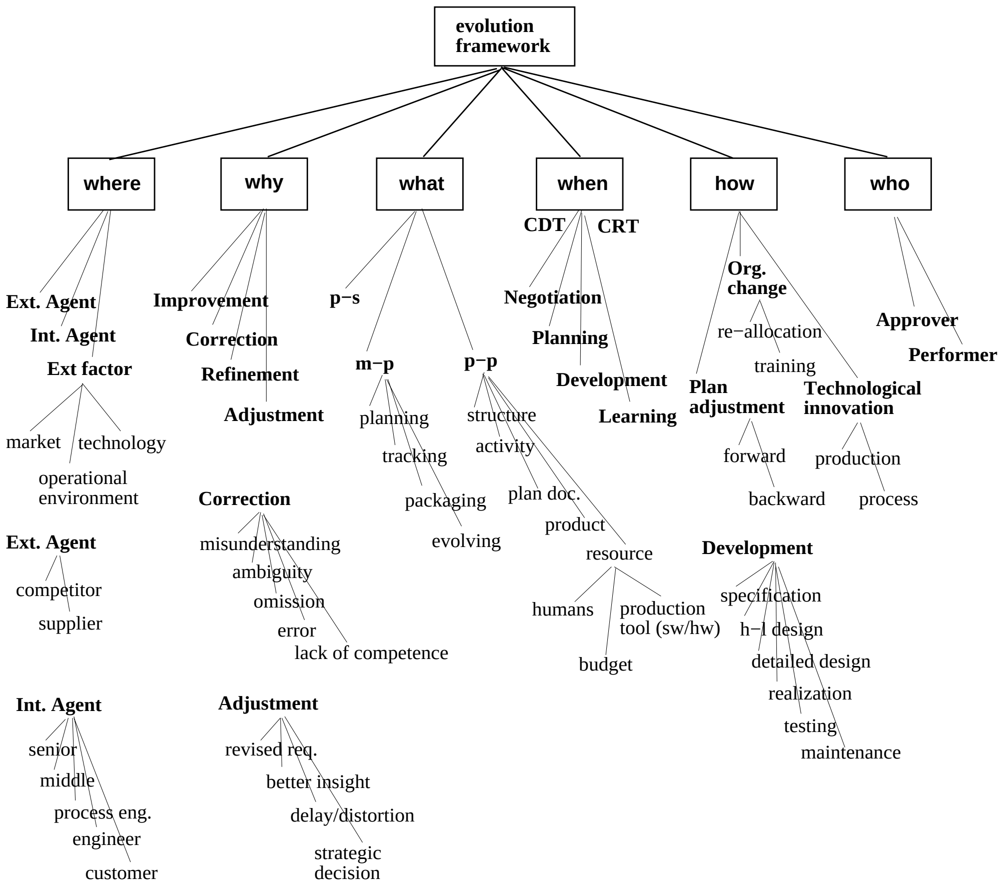


Figure 8.1: Categorization framework for process evolution

and Process Engine [JC93] [LC93]. We have built the EPOSDB [Mun93] to store versioned software products, as well as their related process models. EPOS also supports cooperative transactions.

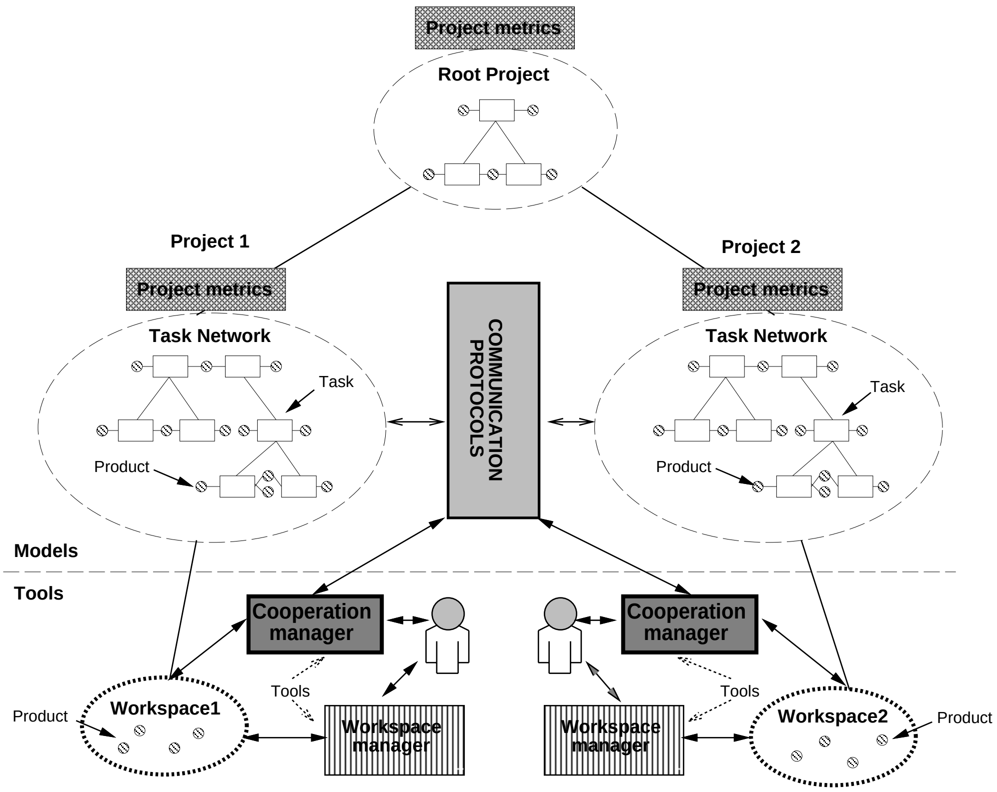


Figure 8.2: EPOS models and tools

Figure 8.2 shows EPOS activity models and tools. The **Process Tools** box shown in figure 8.2, comprises the following EPOS tools:

- **Process Engine**: Used to execute the task-network instantiated by the Planner.
- **Planner**: Incrementally invoked by the `Process Engine` to decompose high-level tasks into an task-network.
- **Task Network Editor**: Makes it possible to directly manipulate the task-network before and during execution, and supports features as add/remove/move Tasks and Products.

- **Project Manager**: Used to plan, start and stop a project and to retrieve useful project metrics from the EPOS-database.
- **Schema Manager**: Responsible for textually/graphically browsing, editing, defining, analyzing, translating and evolving the Process Schema, and can be used on all the process sub-models.

In figure 8.2 two other tools are shown: The **Workspace Manager** which handles versioning of workspaces and products, and the **Cooperation Manager** which is working close to Workspace Manager and handles cooperation and coordination between workspaces (not dealt with here).

8.4.2 EPOS meta-process for managing model evolution

Our meta-process consists of four major steps, and is highly inspired by the Quality Improvement Paradigm of Basili [BCR94a]. The steps are MP1: Planning/instantiation; MP2: Enactment/tracking; MP3: Packaging/Assessment and MP4: Evolving/Learning. MP1 and MP2 are reusing project experience, while the other two MP3 and MP4 are synthesizing such experiences. Figure 8.3 depicts these meta-process steps within a project and associated process tools in EPOS.

8.4.3 Mechanisms for managing process evolution in EPOS

To manage evolution of a software process, it is necessary to store and utilize information characterizing a project and information about changes that are made to the project. It it also important to have PSEE-tool support for changes made to the process model and instantiated process representation. The two next subsections will describe experience database support in EPOS and how EPOS support process model manipulation. The three following subsections describe how to change the instantiated process representation on the fly.

Retrieval of project experience

Project experience is stored in the Experience database as an evolution pattern and associated with a particular project and product profile. Two **characterization** forms have been made to retrieve project and product profiles of the new project. Each characteristic is instrumented by a corporate-specific *weight*. This weight indicates the importance or dominance of a given characteristic when the degree of similarity is determined during selection process of baseline project. Information from the filled-out forms is used to select candidates among completed projects from the database. The baseline project is chosen by summarizing the weights of matching characteristics. The evolution data of the baseline project is available for planning and comparison.

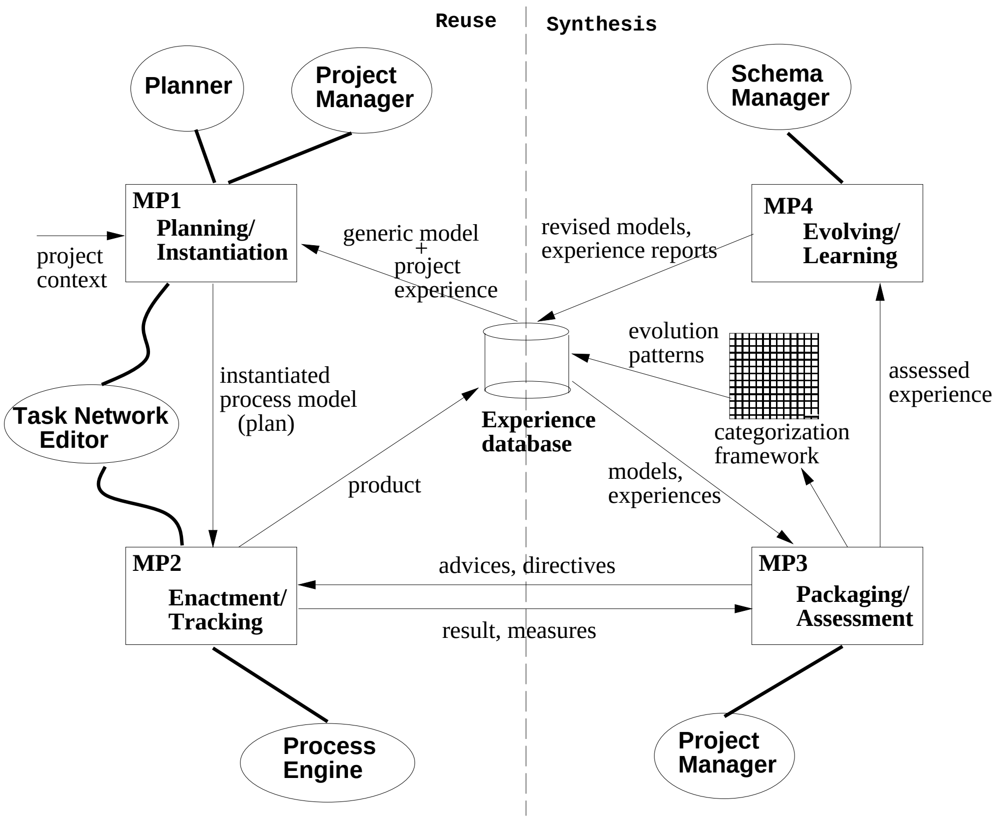


Figure 8.3: Method and tool support for managing process evolution in EPOS

Recording of project experience

While executing a project, changes are recorded by filling out an **evolution request** form. This form collects data sufficient to categorize a given change into an evolution pattern which is stored back to the Experience database with its impact in term of cost. The project performance and product quality measures are also recorded for future learning. On the other hand, new experiences may lead to a revised process model and revised corporate profiles. Such support is provided by the Schema Manager tool which is a tool for changing the process model.

Manipulation of task-network layout

The following techniques can be used to edit the task-network (making changes to the instantiated process representation during enactment):

- **Delete-task**: Remove an task from the task network. This operation can only be done if neighbor tasks can be coupled together.
- **Add-task**: Input an additional task to the task network. This operation can not be applied to the task-network, if it leads to an inconsistent task-network.

By using combinations of the two above you get:

- **Split-task**: Replace one task with two or more new tasks placed side by side. The new tasks are a subset of the task that was replaced.
- **Merge-task**: Replace two or more tasks by one new task. The new task is a collection of all tasks replaced.

Manipulation of task-network scheduling

It is sometimes required to modify the schedule of the task-network to adapt to various change incidents. Such incidents vary from need to revise initial effort estimates to a re-execute or to postpone a start date of a particular task.

During execution on an task-network, we can change individual properties of tasks, such as:

- **Start time** or **stop time** of the task.
- **Allocated time quota** for the task.

Manipulation of task-network resourcing

It is sometimes required to reallocate human resources in order to manage an evolution incident. Thus, it is necessary to support a change of:

- **Human role** responsible for conducting the task.

8.5 Case Study

This section presents the background and context of the case study, and the project profiles and evolution status of the involved projects.

8.5.1 Background and Context

The studied software organization XXX is a Norwegian banking software house with 326 employees of which 89 are working with software production (a typical SME). It is located in three different sites, with the biggest development department in Trondheim. The proposed case study is carried out closely with a System Development Division in Trondheim. The organization is ISO-9000 certified in January 1995 and thus has a documented quality system. Reporting and tracking procedures are installed to collect project performance metrics such as consumed effort, remaining time and cost on paper-based forms. Still, XXX suffers from late deliveries due to inaccurate estimates. It has also realized that unanticipated changes during project is a major cause. Moreover, there exists neither an empirical base nor a quantitative foundation to assess the possible effects of different types of changes based on previous experience. Most projects running at XXX are homogeneous and similar with respect to contractual conditions, product characteristic, solution architecture with operating platforms, customer profile. In addition, projects adhere to a defined project model which includes contract negotiation, planning, water-fall life cycle (i.e. analysis, design, implement, testing) and experience packaging. The project staff is often synthesized from different divisions, depending on the project's required competence/expertise. As the final product is delivered, a corresponding maintenance environment must also be established to assure continuing on-line operation and service. The studied organization needs to have a more sophisticated insight into the evolution profile. Appropriate method and tool support can then be developed. That is the primary objective of the cooperation with our research group. A simulated project environment will be modeled in EPOS and necessary tool support is provided to manage the actual evolution. Lesson learned from the prototyping scenario will be useful for XXX in improving ability in project management.

8.5.2 Project Profile

This Case study is based on five projects named A to E. Following information are retrieved at project start by filling in a project characterization form. Those measures are thus essentially fetched from the XXX's project plan and start report of the studied project. The project profile measures presented in the table 8.1 below are only a subset of those included in the project characterization form used to gather this information. We only select those project profile measures which demonstrate obvious differences between the five studied projects.

	A	**B**	**C**	**D**	**E**
Project type	Develop.	Upgrade	Develop.	Upgrade	Upgrade
Project member	15	14	Varied	8	Varied
Effort (hour)	6990	3815	1150	1500	1520
Duration (mth)	13	9	5	4	4
Customer	Bank	Bank	Bank	Bank	Internal
Competence	Partly	Internal	Partly	Internal	Internal
Technology	Partly	Known	Partly	Known	Known
Difficulty	Manag.	Manag.	High	Middle	Manag.

Table 8.1: Project Profiles for five studied projects

We see from the table that most common project types and sizes at XXX are fairly represented in the project sample. In addition, the degree of risk level which is represented by competence availability, knowledge of applied technology and project difficulty, also cover the entire range of value domains. Projects C and E do not have a fixed number of participants, and personnel are constantly changing during the project life time. That is why their number of project members is defined as *varied*. Product profile is not included in the table because they are partly not of relevance in our further analysis, and partly lacking in the project archive.

8.5.3 Evolution Status

In this section, we present empirical results on evolution which are collected during the case study. A thorough analysis of such data is described in next section. Most information on process evolution are retrieved from from the monthly status reports of the five studied projects. Table 8.2 illustrates the total number of evolution occurrences and their associated correction effort. The average frequency of evolution occurrences per month and average cost for each evolution are then derived.

Due to lack of evolution data in the project archive, some slots in the table above must be left empty. Complete information on process evolution during project E, and evolution cost in projects C and D are not documented at all. Despite of scarce information, the

	A	B	C	D	E	Mean
Total no. of evolution	77	19	17	9	4	31
No. of evolution per month	2.6	1.2	3.0	2.3	-	**2.3**
Total correction cost (hour)	11773	504	-	-	-	6139
Average cost per evol. (hour)	153	27	-	-	-	**90**

Table 8.2: Evolution Profiles for five studied projects

empirical results above demonstrate an average frequency of evolution occurrences of **2 or 3** per month. An average cost for correction effort is about **90** hours per process evolution. Only projects A and B have fully documented the impacts of process evolution. Therefore, we can hardly make further analysis based on such weak data foundation. However, we can demonstrate where process evolution comes from, i.e. where-dimension in the categorization framework. Table 8.3 presents the percentage of different evolution origins from five studied projects and their, average values. The average numbers from the table obviously indicates a superior dominance in percentage of evolution which comes from customer. This conclusion is correct with respect to the perception at XXX. Only now, we have established a quantitative indication to where process evolution comes from.

	A	B	C	D	E	Mean
Customer	34.9	73.6	47	50	50	51,0
Sub-contractor	4.8	0	0	0	0	1,2
Executive	7.2	5.3	17.7	0	25	7.3
Project manager	33.7	15.8	5.9	20	25	19.2
Project member	19.3	5.3	29.5	30	0	21.2
Total	100	100	100	100	100	100

Table 8.3: Percentage of Evolution Origin from five projects

8.6 Evolution Analysis

In this section, we describe the results which are obtained by analyzing collected process evolution data. The analysis is basically performed within the context of our categorization framework (as shown in figure 8.1) . A set of typical process evolution patterns are then identified. Empirical relations between process evolution and project profile are also presented.

We introduce below two ways to perform analysis by combining arbitrary evolution dimensions in the categorization framework (i.e. where, why, what, how, when and by-whom). The following analysis are done by combining dimensions *where-why* and *where-how*. The reason for our choice of such combinations is simply that the analysis results demonstrate interesting findings, and are further discussed in subsequent subsections. Of course, there is no restriction on which or on number of evolution dimensions to be combined to perform the analysis.

The numbers which are reported in following tables, are derived from the EPOS Evolution Analyzer Tool (not described in this paper). Only four greatest process evolution origins in table 8.3 (i.e. Customer, Executive, Project Manager and Project Member) are taken into consideration in the analysis.

8.6.1 Where-Why Frequency Analysis

The percentages in table 8.4 are derived by keeping one Where-category fixed (e.g. Customer), and then varying the category values in Why-dimension. That is, they illustrate percentages of evolution occurrences distributed over different cause categories. Only interesting categories in the Why-dimension are selected for illustration. The most frequent evolution patterns are identified and discussed in depth later in this paper.

	A	**B**	**C**	**D**	**E**	**Mean**
Customer						
Ambiguity	6.5	5.3	17.7	11.1	0	**8.1**
Error	1.3	5.3	0	0	0	**1.3**
Requirement rev.	5.2	15.8	17.7	44.4	25	**21.6**
Lack of competence	3.9	5.3	0	0	0	**1.8**
Delay	15.6	10.5	5.9	11.1	25	**14**
Postponement	5.2	21	0	0	0	**5.2**
Lack of resource	1.3	0	5.8	0	0	**1.4**
Executive						
Re-prioritizing	7.8	5.3	17.7	0	25	**11.2**
Project manager						
Under-estimate	29.9	15.8	5.9	11.1	25	**17.5**
Project member						
Error	11.7	0	17.6	11.1	0	**8**
Lack of competence	5.2	5.3	5.9	0	0	**3.2**
Delay	3.9	0	0	11.1	0	**3**

Table 8.4: Frequency of Evolution Occurrences distributed over Causes

8.6.2 Where-How Frequency Analysis

On the similar manner, the percentages in table 8.5 are derived by keeping one Where-category fixed (e.g. Customer), and then varying category values in the How-dimension. That is, they represent the percentages of evolution occurrences distributed over different impact categories. Only some interesting categories in the How-dimension are included in the table. The identified impacts are used to validate EPOS' approaches to manage process evolution in section *Managing typical evolution patterns in EPOS.*

	A	B	C	D	E	Mean
Customer						
Rework	7.8	15.8	0	11.1	25	**11.9**
Prolong task	7.8	31.8	23.5	33.3	0	**19.3**
Postpone task	20.8	21	11.8	0	25	**15.7**
Revise/re-estimate	1.3	5.3	11.8	11.1	0	**5.9**
Project manager						
Rework	11.7	0	0	0	0	**2.3**
Prolong task	7.8	15.8	5.9	11.1	0	**8.1**
Project member						
Rework	7.8	0	11.8	22.2	0	**8.4**
Prolong task	9.1	0	23.5	11.1	0	**8.7**
Postpone task	10.4	0	5.9	0	0	**3.2**
Training	1.3	5.3	5.9	0	0	**2.5**

Table 8.5: Frequency of Evolution Occurrences distributed over Impacts

The table 8.5 shows that rework and prolong task are superior evolution impacts on project schedule. Such impacts require additional allocation of resources according to initial plan. The frequency of evolution which requires rework, constitute 23% (summarizing the Rework numbers from table 8.5), while 36% of total evolution occurrences imply additional resources allocated to a particular task (summarizing the Prolong tasks numbers from table 8.5). A postponed task due to a process evolution does not involve any additional cost. However, such evolution imply that the plan must be re-scheduled and allocated resources must be released. Still depending on the extent of resource needed to such rescue tasks, the figures in the table clearly demonstrates the importance and seriousness of process evolution.

8.6.3 Typical Evolution Patterns

Based on the figures presented in tables 8.4 and 8.5 above, we identify eight most typical and frequent evolution patterns. Such evolution patterns are below listed according to decreasing frequency percentage. Within the description of evolution pattern, we also emphasize the necessary corrective actions. Such rescue actions will be then realized by invoking different operations to manipulate project plan (i.e. task-network) in EPOS (see section *Mechanisms for managing process evolution in EPOS*).

1. **Customer revision (21.6%):** The customer issues new or revised requirements according to the initial specification. Such changes requires that completed tasks must be re-executed. The extent of rework depends on which phase in development life-cycle the requirement change is requested. New tasks can also be added into the project plan to deal with new or enhanced system functionalities. Note that the requirement revisions after initial delivery (i.e. maintenance phase) have not been taken into consideration in this case study.

2. **Under-estimation (17.5%):** The project manager tends to under-estimate various tasks under planning. This is partly due to misunderstanding of user requirements, and partly lack of empirical foundation for making estimates. Such under-estimation usually causes that initial estimates must be revised, and project plan must be rescheduled. Sometimes, new tasks are introduced while existing ones are either removed, divided or merged.

3. **Customer delay (14%):** The customer delays to deliver documents (e.g., requirement, customer site interface) as initially agreed. Such delays require a postponement of tasks which are dependent on the given deliverables. The project plan must then be revised to reflect that fact.

4. **Resource re-allocation (11.2%):** The executive management (division or senior manager) re-allocates project personnel to other project as a result of strategic re-prioritizing. Such happen when they want to "rescue" other urgent projects, or to satisfy a higher prioritized customer request. This high frequency of staff turn-over results in a violation of the project plan. That is, necessary resources and competence are not available to keep in project progress on schedule. Human factors associated to a particular task must be removed, and other dependent tasks must therefore be postponed.

5. **Ambiguous requirement (8.1%):** The customer is not clear when the initial requirements are specified. This is due partly to their limited insight into the problem, and partly due to their lack of competence in a given technology. That is, important requirement details are neglected or vaguely defined. Such ambiguous requirements usually imply a rework of requirement specification, or an insertion of corrective tasks.

6. **Error commitment (8.0%):** The project member have injected errors in technical documents. Such an evolution pattern often initiates either rework or prolong the current task. An specialized error removal task can also be inserted to deal with the problem. Such corrective actions imply delay and thus re-scheduling of the project plan.

7. Customer postponement (5.2%): Unexpected requests for postponing the start date of an task are issued by the customer. Such postponement causes that the initial project plan must be revised. The time and human factors which are associated to scheduled tasks are changed.

8. Lack of competence (5.0%): This phenomena is more common among project member (3.2%) than among customer (1.4%). Such events are resolved by assigning project personnel to training to improve e.g. their knowledge on a particular development method/language. Training tasks are thus added to the existing project plan.

Based on the identified evolution patterns above, we have suggested a set of improvement initiatives which are described in the section *Managing typical evolution patternsin EPOS*.

8.6.4 Empirical Relations between Evolution and Project Profile

In this section, we present our findings on empirical relations between collected data (in previous section) and project characteristics (see table 8.1). In particular, we observe typical evolution patterns on one hand, and external project profiles on the other hand. Such empirical relations improve our ability to predict and then anticipate process evolution based on what we know at project start (i.e. project profile). We have revealed following interesting relations. Of course, this list is not exhaustive.

Project type vs. Evolution frequency Development projects (A and C) experience more turbulence than upgrading ones (projects B, D, and E). This fact is illustrated by the high percentages of evolution frequency in table 8.2 in projects A and C. This phenomena can be explained by the fact that both customer and XXX in development projects deal with a new problem which they often do not have sufficient technical competence.

Project type vs. Customer revision A high degree of customer revisions is found in upgrading projects (B, D, E) in table 8.4. This observation is somehow hard to explain. There are two assumptions usually associated to upgrading projects. Firstly, technology is well-known, and required competence is available. Secondly, the customer often states clear and well-defined enhancement requirements as they have been using the product for a period of time. However, the second assumption does not hold in our case. For instance in project B, the customer at project start delivered an requirement specification which has thereafter been revised so many times that the initially agreed-upon contract must be completely re-negotiated. Therefore, this empirical relation must be validated carefully later as with more data points.

Project duration vs. Under-estimation The degree of under-estimation seems to increase proportionally with the project duration (see table 8.4). It is easy to accept the fact that it is hard to predict the behavior of some tasks taking place far beyond in the future. Moreover, project managers at XXX suffer from the lack of empirical baselines from prior projects to rely their estimates on.

Project duration vs. Customer delay The extent of customer delay is proportional to the project duration (see table 8.4). This is probably due to the fact that customer tends to give such projects lower priority, and thus does not respect the initially agreed-upon deadlines.

Project type vs. Ambiguous requirement Customers in development projects are often uncertain on what features they really need. This normally leads to unclear and vague specification of initial user requirements.

Project type vs. Error commitment Table 8.4 shows a superior representation of errors done by project members in development projects. This is perfectly natural when the project members must address a problem to which they do not completely master.

Customer type vs. Customer revision Development project E with an internal customer (i.e. XXX's Product Division in this case study) has experienced a higher degree of requirement revisions than other projects. This relation is not quantitatively illustrated by the collected data material, but has been acquired through reading other reports. Nevertheless, it is easy to understand when we accept the fact that internal development projects do not bear any economical risk for XXX, and in addition are often given lower priority by the executive management level.

It is worthwhile to stress that the empirical relations above are retrieved by our limited data ground which are collected during the case study. The explanations for the relations are thus somehow speculative. As the amount of collected data grows, more valid relations can then be extracted. After all, our primary intention is to demonstrate the usefulness and benefits from collecting and analyzing such information.

8.7 Managing Typical Evolution Patterns in EPOS

To deal with the typical evolution patters necessary actions must be made to correct a particular evolution pattern. Most of them imply a revision of the project plan (i.e. task-network in EPOS) by either adding new tasks, manipulating existing ones, changing schedule properties of tasks.

Below we summarize basic operations that can be used to manipulate the EPOS task-network:

Adding task Put an additional task to the existing task network.

Deleting task Remove an existing task from the task network.

Schedule revision `Start` time, `Elapse` time and duration of the task in the network can be revised during execution.

Resource revision Performer of the task in the network can be replaced. As the consequence, the task is either suspended or canceled.

Table 8.6 shows how eight evolution patterns (enumerated 1–8 according to the list presented in last section) are dealt by EPOS' approaches to manage task-network evolution.

Evolution pattern	1	2	3	4	5	6	7	8
Delete task operation							x	
Add task operation	x				x	x		x
Task schedule revision	x	x	x	x	x	x	x	x
Task resource revision	x			x	x	x		

Table 8.6: Evolution Patterns vs. Approaches in EPOS

8.8 Improvement Suggestions

In this section we state our suggestions for improvements towards the studied software company XXX on the basis of case study results and findings.

8.8.1 Suggested Improvement Initiatives for XXX

By consolidating the findings and analytical results from the case study, we propose following suggested improvement initiatives for XXX in the future. The following list is not sorted in any priority order.

- The most frequent evolution pattern is revealed to be user requirement revision. XXX should have therefore better dialog with customer in requirement specification phase. That is, advanced technology in requirement engineering can be adopted to better understand requirements and thus prevent them from modifying later.
- The second most frequent evolution pattern is concerned the issue of under-estimation. This problem can be remedied adopting better estimation method. The case study has demonstrated the benefit of learning from past experiences to establish baselines and then improve estimate accuracy. The existing quantitative baselines at XXX are still insufficient and incapable to provide precise estimates. However, they will be improved over time if XXX commits to pursuit the the same work direction. On the

other hand, XXX should initiate better cooperation between System Development Division and Sale Division in contract biding process. That is, realistic but also competitive bids must be elaborated together.

- Incorporate new paper-based forms for characterizing project/product; collecting evolution data; collecting actual project/product status and categorizing errors into appropriate handbooks in the Quality System. Specifically, the three first forms are included in the Project Management Handbook, while the last one in the System Development Handbook. By doing so, XXX institutionalizes an effective data collection process for the company. Further support for data collection and utilization of collected data, can be provided by introducing an Experience Database (EDB). Since an the Experience Database used needs times to be validated, those paper-based forms can be used in the meantime.

- Select one or two pilot projects which will use the EDB for recording project/product measures and process evolution. In parallel, the collected data from the five studied projects should be stored in EDB available for use. The reason to run EDB together with pilot projects is to validate its applicability, user-friendliness and correctness. Necessary enhancements should be recorded and installed before EDB is released for use within the company.

- Project managers are encouraged to use the EDB both to make project estimates, and to anticipate possible impacts during project execution (planning with contingency). It is a major step towards continuous learning and reusing past experiences. As many projects record their data into the EDB, the existing baselines gradually become valid, representative and convergent. The improvement evidences on estimate accuracy are then significantly demonstrated. Patience and full commitment are key issues to establish a valid experience foundation. We refer to an example of at NASA-SEL. It took them 18 years (1977-94) with several experiments to gather data before they manage to consolidate to a set of representative and reliable baselines.

- To be able to achieve significant improvement in error reduction during development, XXX is suggested to seriously collect and classify errors according to their types, origins, severities, and related characteristics (partly covered by the an error report form). XXX can then obtain a deep insight into the error problem and prioritize corrective actions according to error impacts. Adequate technology can thus be evaluated and introduced to reduce number of errors or to detect them earlier. That is, required effort to remove them can be declined considerably.

8.9 Conclusion

In this paper we have presented a taxonomy for classifying process evolution along six dimensions. A case study has been conducted with a software organization to collect actual evolution data based on the defined categorization framework. The preliminary analysis shows that most changes originate from customer delays or requirement changes. Such changes are detected in the design and testing phase, and constitute a major impact in term of extra cost. A QIP-inspired method with an experience database and associated tools have been developed in EPOS to manage and analyze collected evolution occurrences observed changes. Such support will improve the ability for planning and scheduling to reduce unanticipated changes during project. The gained experiences is fed back to the organization for making effective improvement decisions. However, the reuse of previous project experiences has been negligible since such evolution data are non-existing. More projects will be tracked to make the experience database more confident and complete.

Further, much research effort has dealt with managing changes on process models in term of evolving types or classes. From our new work, we realize that rather pragmatic changes to task instances (instantiated process) are much more frequent than type evolution. The consistency impacts of such instance-level changes need to be better managed and deserve more research attention.

CHAPTER 9

Planning Support to Software Process Evolution

Reidar Conradi, **Minh Ngoc Nguyen**, **Alf Inge Wang**[1], and **Chunnian Liu**[2]

Abstract
The ability to handle changes is a characteristic feature of successful software projects. The problem addressed in this paper is what should be done in project planning and iterative replanning so that the project can react effectively to changes. Thus the work presents research results in software engineering, as well as transfer of methods in knowledge engineering to software engineering, applying the AI planning technique to software process modeling and software project management. Our method is based on *inter-project experience* and *evolution patterns*. We propose a new classification of software projects, identifying and characterizing ten software process evolution patterns and link them to different project profile. Based on the evolution patterns, we discuss the planning support for process evolution and propose several methods that are new or significantly extend existing work, e.g. **cost estimation of process changes, evolution pattern analysis**, and **a coarse process model for the initial planning- and the iterative replanning process**. The preliminary results have shown that the study of evolution patterns, based on inter-project experience, can provide valuable guidance in software process understanding and improvement.

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway. Phone: +47 73 593444, Fax: +47 73 594466, Email: conradi/nguyen/alfw@idi.ntnu.no

[2]Beijing Polytechnic University (BPU),
Beijing, P.R. China.
Chunnian Liu's work is partly supported by the Natural Science Foundation of China (NSFC), Beijing Municipal Natural Science Foundation (BMNSF) and Chinese 863 High-Tech Program.

Keywords: *Software Process Modeling, Software Process Evolution, measurements, planning.*

9.1 Introduction

Coarsely, we have four kinds of changes/evolution in software processes. First, we have two kinds of changes in a normal development process, namely delayed planning (e.g. of design details), or replanning due to unforeseen events (e.g. changes in requirements or in staffing). Due to the unstructured nature of many of these changes, traditional project management tools are grossly insufficient [Håk94]. A third kind of change occur in a later maintenance process, where a stream of change requests are being handled. These changes can be characterized as perfective (50%), corrective (21%) adaptive (25%) and preventive changes (4%) [LST78]. Lastly, a fourth kind of change is the more long-term *software process improvement (SPI)*, which implies goal-driven and systematic evolution of the entire software process between projects, cf. CMM [PWCC95] and Experience Factory [BCM$^+$92].

So the scope of research in software process evolution is vast. The particular problem addressed in this paper is what should be done in project planning so that the project can react rationally and effectively to changes. Thus, we focus on the first two kinds of changes, with some emphasis on the fourth (i.e. SPI).

We propose a framework of project profiles, process evolution patterns observed in projects, and the relation between project profiles and evolution patterns. Then we discuss the planning phase in depth, aiming to provide the project manager with method support in configuring a more realistic and flexible project plan. This includes initial planning before the start of the project and iterative replanning during the execution of the project. The method is based on inter-project experience and the evolution patterns. For a new project, a matching baseline project is selected from an Experience Database to predict the potential change patterns that are likely to occur in the new project. Based on this information, the project manager can make more appropriate decisions in the project plan so that the project is better prepared for the potential changes. These decisions will be followed and/or revised, when changes do occur during the execution of the project. We also present some preliminary, experimental results - including an industrial case study - which covers part of this paper.

This work presents not only research results in software engineering alone, but also transfers methods of knowledge engineering to software engineering, applying the AI planning technique to software process modeling and software project management.

9.2 Related Work

Lehman [Leh94] has presented a classification of software projects and process evolution, listing five evolution entities: Software Releases, Software Systems Under Development, Application Domains, Development Process, and the Process Model. His paper also describes the evolutionary characteristics of each of these entities. However, his discussion is in a very abstract manner.

On the other hand, Nguyen and Conradi propose a functional framework (neither too abstract, nor overly complex in operational details) to classify process evolution and link evolution patterns to different project profiles [NC94b]. The evolution characteristics are: where (origin), why (cause), what (process element), when (phase), how (kind of change), by-whom (modifier). All these characteristics can be subclassed.

The original EPOS Planner [Liu91] exploits a mixture of hierarchical and linear planning [AIS88] to generate parts of the task network in a process model. In [LC93], Liu and Conradi present an incremental replanning algorithm to deal with “microscopic” changes in the process model. For example, the replanning algorithm can adjust an existing task network to reflect changes in the Product Structure and/or in task types (called the template process model). But in real processes most changes with major impact on the process quality and improvement are “macroscopic”; see [NWC97]. Our previous work on replanning was also confined to individual projects, and inter-project experience on process evolution was not considered. The (re)planning method presented here is based on inter-project experience and more general evolution patterns.

In the domain of *software experience databases*, we have mentioned the Experience Factory improvement method developed at NASA-SEL and Univ. Maryland. The Experience Factory approach proposes a centralized improvement taskforce in a company, with long-term and company-specific SPI as a goal, and is internally exploiting an experience database. There is also the associated TAME project [BR91] to exploit reusable process artifacts in an experience database, e.g. to make decisions about subsequent projects within the same organization. Further, much work on experience databases and corporate memory as a vehicle for for organizational learning has been done, especially in the engineering sector. This has often been accompanied by techniques such as statistical analysis, data mining like that provided by rough-sets [Paw91], or AI-oriented methods like case-based reasoning [Lea96]. Finally, there is also work on the use of OO-style patterns to represent reusable classes of projects [JGJ97].

Madachy has implemented a knowledge-based system [Mad95] for risk assessment and cost estimation. More recently Konito in [Kon96] proposes a systematic method for risk management, where risk is seen as variance or degree of imprecision. The idea of risk scenarios (the hierarchy of risk factor – risk event – reaction) is adapted in this paper to systematically make planning decisions about how to deal with changes which are likely to occur in the project; see Section 9.4.3.

9.3 Classification of Process Evolution

9.3.1 Classification of Projects

First we observe that it is not realistic to talk about evolution patterns for software projects in general without linking them to different profiles of projects. Typical *overall-profiles* may cover aspects such as *cost-critical*, *time-critical*, *safety-critical*, *research-oriented* etc. Figure 9.1 shows a top-level classification of software projects, which we call *overall-profiles*, in which a list of items (*purpose, application domain*, *software product, main concerns, typical changes, main risk*) are used to characterize the overall profiles.

	A	**B**	**C**	**D**
Projects *descr.* *Charact.*	**classical time/budget constrained projects**	**time−critical projects**	**safety−critical projects**	**research prototyping projects**
purpose	develop/maintain application software in a particular domain	occupy market of a public software	develop/maintain applic. software in safety−critical domains	test new concepts and techniques, using extensive prototyping
application domain	commerce, industry, medicine, communication, ...	public sector etc.	transportation, military and space, nuclear, real−time control, ...	science, computer science, software technology, process improvement
software product	application software tailored software	system software (languages, OS, DBMS, Network,......)	application software tailored software	prototype software to test/validate new ideas, algorithms, methods, ...
main concerns	run the project within budget, deliver software in time, user satisfaction	put the product to market as soon as possible	no critical errors, remaining errors are as few as possible	validity of new methods and comparison with existing ones
typical changes	external: requirements, resource internal: review/testing	external: market internal: requirement, design	external: error report internal: testing	internal: specification
main risk	schedule delayed, budget overrun, user complaints	release delays	critical errors causing serious losses of life/money	very low

Figure 9.1: Classification of Project Overall Profiles.

We do not claim that Figure 9.1 shows a complete list. Also, sub-classification is possible and may be necessary. For example, each of overall profiles A-D in Figure 9.1 can be further classified according to the following characteristics among others:

Application domain: Telecom, Business-Processing, Military, MIS, Banking, *Novelty*: release *vs.* new software; familiar *vs.* new application domain, and *Software maturity level of the organization*.

In addition to the overall-profile of projects, we have an *operative-profile* with categories such as: *project manager, participants, contractual-constraints, project size, external-suppliers, customers, operating-platform, implementation-languages, etc.* Each part of the operative-profile is assigned a corporate-specific *weight*, representing the relative importance of the part in determining the similarity between various projects of the corporate. For example, a software house XXX may assign a high weight on *contractual-constraints*, as it learned that projects with matching constraints behave similarly.

When we speak of project profile, we mean both overall- and operative-profile. In the following, we assume that the Experience Database has been initialized with data from a non-trivial number (20-50) of historical projects, according to the profile classification described above. Linked to these project descriptions, we have attached template process models that have been used in these projects. Such templates constitute a repertoire of reusable process model parts that can be combined and instantiated into operative process/project models. In such an Experience Database we also store estimation/risk models for project planning, and quality models e.g. about defect densities.

9.3.2 Evolution Patterns and their Relation to Project Profile

In our approach, a *evolution pattern* is recognized by the following five steps:

1. Set up a framework with a hierarchy of characteristics describing various aspects of change. In this paper we use the following top-level attributes, being an extension of those in Minh's PhD thesis [Ngu97] (Figure 9.2):

    - *Source of change*: internal (triggered by feedback), or external (triggered by change of input).
    - *Scope of change*: local (microscopic), or global (macroscopic).
    - *Timing of change*: in early or late phases of the process.
    - *Evolving entity*: application-domain /environment /product /process /process-model.
    - *Forward-action*: actions taken in the forward path of the process.
    - *Backward-action*: actions taken in the backward loop (feedback loop).
    - *Impact on schedule etc.*: big/moderate/small impact on schedule/quality/cost.

2. Keep track of change occurrences over a large number of projects in an organization, record them in a form complying with the framework, and store the data into an Experience Database of the organization.

3. Set up criteria to classify changes defined by the project manager. Here two recorded changes with the same attribute values are considered "identical."

4. Retrieve and analyze the evolution data stored in the Experience Database, grouping similar changes with only "small differences" according to some criteria.

5. Each group of *similar changes* with significant number of change occurrences (say, more than 5% of the total number of changes being analyzed) can be seen as an *Evolution Pattern*.

Figure 9.2 shows some common evolution patterns. For each pattern we give the "normal" value for each of the attributes. Each evolution pattern contains the following information: The *occurrence frequency* of a certain change in the project; The *explicit/implicit reactions* taken to deal with the change; The *impact* of the change on schedule, cost and quality (the risks); And the associated *metrics* (measuring procedure, data attributes, analysis model and analysis results). Naturally, such evolution patterns are also recorded in the Experience Database, and their usage are described in Section 9.4 on planning.

The relation between project profile and evolution patterns could be extracted from the Experience Database, or a model of this relation could be validated/invalidated by the data in the database. Figure 9.3 shows an example of such a relation, linking project overall-profiles and evolution patterns.

9.4 Planning Support for Software Process Evolution

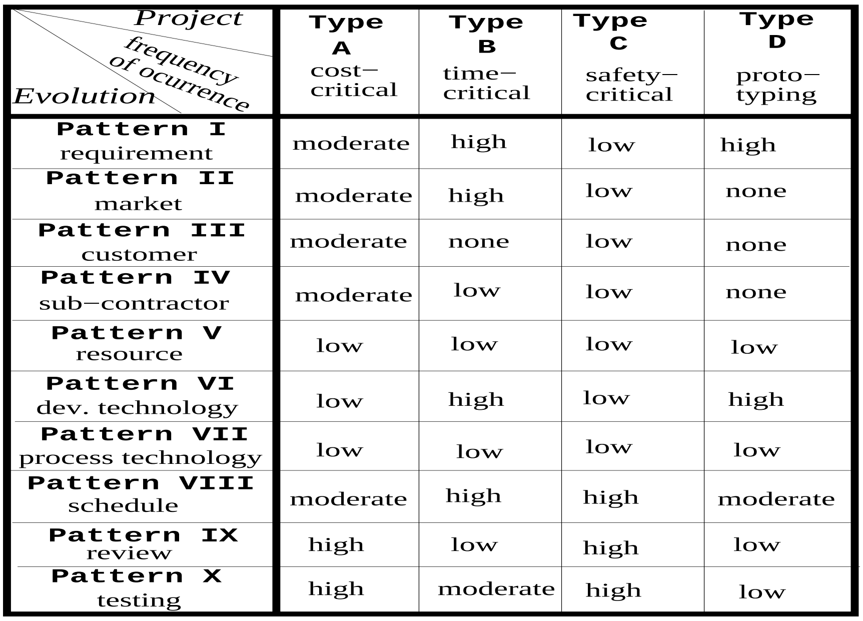
Our method is based on *inter-project experience* and *evolution patterns*.

Each previous project will have its overall/operative-profile recorded in the Experience Base (EB). As we mentioned in Section 9.3.1, items in the operative profile include: *project manager, participants, contractual-constraints, project size, external-suppliers, customers, operating-platform, implementation-languages, etc.*, and each item of the profile is assigned a corporate-specific *weight* to express its importance in measuring the similarity between projects.

A new project is initiated with its own overall/operative- profile. The project manager uses it as the criteria to search the Experience Base for those previously completed projects that are similar to the current project. Project similarity can be discovered by partial matching between the corresponding profile items. These similar projects constitute the candidates to be considered as the baseline for the current project. The best-matching candidate, *i.e.* having the highest sum of the *weighted* attributes associated with the matched measures, is then selected as the baseline project. Of course in practice, the choice of the baseline

Charact / *Values* / *Patterns*	**Source of change**	**Scope of change**	**Timing of change**	**Evolving entity**	**Forward-action**	**Backward-action (feedback)**	**impact on schedule,cost, product quality etc.**
Pattern I requirement change	external	macro	early phase (usually)	appl.domain, product, process	re−schedule forward path	re−analysis, re−design, re−implem.	early phase: moderate later phase: big
Pattern II market change	external	macro	any time	environment, product	re−schedule forward path	re−analysis, re−design, re−implem.	early phase: moderate later phase: big
Pattern III customer delay	external	macro or micro	early phase (usually)	environment, process	re−schedule forward path	none	risk to schedule: depends
Pattern IV sub−contractor delay	external	macro or micro	later phase of process	environment, process	re−schedule forward path	none	risk to schedule: depends
Pattern V resource re−allocation	external (internal to organ.)	macro	any time	environment, process	re−schedule forward path	none	risk to schedule: big
Pattern VI new development technology infusion	internal	micro	late phase	process, process−model	improvement in forward path	negative feedback to constraint	global impact is constrained by negative feedback
Pattern VII new process technology infusion	internal	macro	early phase	product, process, process−model	improvement in forward path	negative feedback to constraint	global impact is constrained by negative feedback
Pattern VIII internal schedule adjustment	internal	macro or micro	any time	process	re−schedule forward path	none	moderate
Pattern IX review−modify cycle	internal	micro	early phase	product, process	wait till termination of the cycle	re−design, re−analysis	moderate
Pattern X test−modify cycle	internal	micro	late phase	product, process	wait till termination of the cycle	re−implem., re−design, re−analysis	moderate

Figure 9.2: Classification of Evolution Patterns.



Project / frequency of ocurrence / Evolution	Type A cost–critical	Type B time–critical	Type C safety–critical	Type D proto–typing
Pattern I requirement	moderate	high	low	high
Pattern II market	moderate	high	low	none
Pattern III customer	moderate	none	low	none
Pattern IV sub–contractor	moderate	low	low	none
Pattern V resource	low	low	low	low
Pattern VI dev. technology	low	high	low	high
Pattern VII process technology	low	low	low	low
Pattern VIII schedule	moderate	high	high	moderate
Pattern IX review	high	low	high	low
Pattern X testing	high	moderate	high	low

Figure 9.3: Relation Between Project Overall-Profiles and Evolution Patterns.

project is eventually a human decision, but the above method can provide some automatic help in the decision-making process.

Furthermore, the evolution patterns of the baseline project, which were recognized using the method given in Section 9.3.2, were also recorded in the Experience Base (EB). Retrieving this information, evolution patterns become predictable for the current project.

In this section, we will discuss how a project manager can utilize the evolution information of the baseline project to configure a more realistic, flexible plan for the current project. He/she is then expected to be better prepared for potential changes, which can be derived or predicted from the baseline project.

Project planning consists of initial planning and iterative replanning activities.

With our approach, the initial project plan will include the following essential items (besides general descriptions of project objectives, constraints, organization, resource, etc.):

1. *Task network*: This expresses the hierarchy of and dependencies between activities. In the initial plan, only a rough network can be provided. This is then refined/modified during the life-time of the project.

2. *Schedule*: The allocation of start/end time and various resources to each activity, and a set of intermediate milestones, (presented by a set of charts).

3. *Cost estimation*: The estimated effort of the whole project, as well as of each activity. In our case, the cost of possible changes should also be estimated.

4. *Evolution pattern analysis*: Based on the anticipated evolution patterns from the baseline project, try to actively change the project profile and/or improve the the process model. If this is not feasible, set appropriate contingencies (extra activities/time/resource) in the project plan to deal with possible changes.

5. *Metrics*: The relevant attributes and associated procedures to collect/analyze them. The actual measures will be used for risk analysis, plan revision, experience learning for future projects, etc.

We first present in Figure 9.4 our coarse process model (as pseudo-code) of the project planning process, significantly extending the one given in [Som95]. The planning process has four steps, S1-S4, where we will focus on step S1 "Initial Planning" and S3 "Replanning." The following subsections will describe our methods for each of the plan items, focusing on the core problem: "How to deal with possible changes". The resulting plan will be "better" than the ones which are based on a particular individual's experience or on some algorithmic models. It is "better" because it is based on the whole organization's experience and is pre-prepared for possible changes.

9.4.1 Task Network Layout and Scheduling

The task network shows the task hierarchy (sub-tasking) and inter-dependencies. Initially, only a rough task network can be created. During project execution (e.g. in the design phase), high-level tasks are gradually decomposed into sub-tasks in a hierarchical way.

The project schedule should fill in start/end time and resource allocation for each task, all milestones representing important stages in the project, and the *critical path* – the longest path in the task network. Scheduling is based on cost estimation which is the subject of the next subsection. The project manager must consider possible scheduling changes. The following guidelines apply:

1. Schedule first as if nothing will go wrong, then increase the time periods and resources for tasks affected by anticipated changes (see Section 9.4.3).

2. Estimates in the initial project schedule must be compared with actual elapsed time in order to revise the schedule for the later parts of the project, even re-partition the later parts to reduce the length of the critical path (see Section 9.4.4).

3. To react to a change during project execution, the layout of the current network may need to be modified (adding/deleting/modifying tasks). In most cases, the modification is a human decision (such as adding an extra review task to enhance reliability), and can only be done manually (with some help of graphical editing tools).

S1. Initial Planning

```
Define project constraints
Estimate initial project profile
Estimate project cost (whole and for each phase)   (Section 4.2)
Define project milestones and deliverables
Generate automatically the top−level task network   (Section 4.1)
Make the initial schedule   (task break−down,  period/resource allocation,  Section 4.1 and 4.2)
Evolution pattern analysis:   (Section 4.3)
        Whenever possible and desirable
            Try to change the project profile and/or the process model, adapted to evolution patterns
        Otherwise
            Put contingencies in the project plan
Make sub−plan for measurements and risk analysis (Section 4.4)
```

S2. Enact the tasks according to schedule

S3. Continuing, iterative, (re)Planning

```
while {project is not completed}
      Task Decomposition:  (Section 4.1)
            Decompose high−level tasks during the execution
            Whenever possible, use the automatic Planner
            Otherwise use Graphic Editor to do so manually
      Tracking and Monitoring:  (Section 4.4)
            Track progress
            Collect and analyze the collected measures according to the measurements sub−plan
      Analyze Risks:   (Section 4.4)
            if    (the risk is high)
            then  revise estimate
                  revise schedule  or
                  re−negotiate project constraints/deliverables
            end if
      Deal with Changes:   (Section 4.3,  4.1 and 4.2)
            if     (a change scenario occurs)
            then   execute the reaction of the scenario
                   record the occurrence of the scenario
            elseif (unexpected changes occur)
            then   deal with it based on the manager's judgement
                   record this new evolution pattern/scenario/occurrence
            end if
end while
```

S4. Experience Packaging and Learning: put back into Experience Database

Figure 9.4: The coarse Process Model of Project (re)Planning Process.

9.4.2 Cost Estimation

Project cost reflects efforts, training, equipment, and so on. Here we only consider the *efforts costs* – the costs of paying project staff according to hours spent and their price.

Initial cost estimation is necessary to negotiate the project budget and the product price, to set the periods and resources for each task in scheduling. Continuous cost recording is later needed to ensure that spending is in line with budget (often being the estimate). Cost estimation for changes is necessary to effectively manage the process evolution.

Our overall method for cost estimation is:

- *Using experience data to help traditional cost estimation*:

  There is extensive literature on this, describing two kernel techniques of software cost estimation: *Estimation by analogy* (often manual) and *Algorithmic cost modeling* (often automatic). Our method is helpful for both. Estimation by analogy is based on completed projects of the organization and in same application domain (the baseline); Algorithmic estimation models such as COCOMO [Boe81] have to be tuned to the need of a user organization using historical project data. In our approach, the baseline project and historical measures are both stored in the Experience Database and available to the manager of the new project.

  As a minimal example, using the COCOMO model, we have the following estimation steps and results for a project implemented in C++:

  1. Calculate the number of *function points* FP, e.g. 150 (estimated in requirement analysis or perhaps in high-level design);
  2. Code size $LOC = AVC^3 * FP = 7500$, where $AVC = 50$ for C++;
  3. $Effort = 2.4 * LOC^{1.6} = 20$ person-months, where constants 2.4 and 1.6 are tuned using historical data, including documentation and testing;
  4. Project duration is $2.5 * Effort^{0.38} = 7.8$ months, i.e. the project will be carried out by a 3 person-team (again, with constants 2.5 and 0.38 are tuned using historical data).

- *Cost estimation of process changes*:

  – The baseline project contains historical cost data for each change pattern. These data can be used to estimate the cost of similar changes in the new project. This method can be applied to the review-modify cycle (`pattern IX`), test-modify cycle (`pattern X`), and other evolution patterns.

  – For a new requirement, COCOMO or other models can also be used to estimate the cost to satisfy the requirement, e.g. expressed as FPs.

[3]AVC = Average LOC for each function point.

  – For requirement modification, the cost depends on the timing of the change occurrence. Modifications occurring in early phase cost little, as the real implementation work has not started yet. Changes occurring in later phases, however, may cause costly profound re-design and re-implementation. Requirements traceability techniques [LS96] can be used to identify the affected software modules to estimate the efforts of such re-work.

9.4.3 Evolution Pattern Analysis

To achieve real SPI, the project manager should analyze carefully the anticipated evolution patterns from the baseline project, and try to actively change the project profile and/or improve the the process model before the project starts. For example:

- If an evolution pattern indicates intrinsically unstable user requirements, we may choose an incremental development-delivery model with “time-boxing” [Red97] to replace e.g. the waterfall model employed in the baseline.
- If an evolution pattern indicates a high defect rate, we may introduce more rigorous inspection tasks in the project plan to reduce the risk in testing phase.
- If an evolution pattern indicates high people turn-over, we need to prioritize the over-crowded assignments in the company’s pipeline (an improvement of the organization and the project profile).

Sometimes it may not be feasible to change the project profile or the process model. In these cases, there should be some contingency (extra time/resources) decisions in the project plan to be better prepared for anticipated changes. For example:

- Contingency factors (the percentage of extra time/resources) should be decided for those tasks which are affected by the expected change patterns.
- Some resources should be reserved for possible re-work to cope with the anticipated change patterns.
- Some possible extra tasks should be perceived beforehand to deal with changes, for example re-negotiation with the customer about budget, functionality, and deliver date in case of later requirement change.

In the following, we propose a systematic method to conduct the evolution pattern analysis and to make corresponding decisions. Several evolution patterns can be expected for a new project. Each pattern could cause different change events, and each event has alternative reactions and impact on the schedule/quality/cost. To provide a control mechanism for analysis and decision making, we can create and use *evolution scenarios*. In the Riskit method [Kon96], Kontio suggests to classify identified risks into elements (factors and events), and describes plausible reactions for each event, resulting in what he terms a risk scenario. However, his method does not utilize inter-project experience. In our case, an evolution scenario is a path from an evolution pattern to one of its events, and further to one possible reaction. The evolution patterns are anticipated from the baseline project. Thus, it is easier and more natural to break down a pattern into several events, and to provide alternative reactions to each event. All these are guided by the information in the evolution pattern which embodies inter-project experience. Another difference between our method and Riskit is that our proposed decisions are mainly made during the planning phase - before a project starts - while risk analysis in Riskit is mainly carried out during the project execution.

Using evolution scenarios, the main steps in decision making become:

1. For each evolution pattern, perceive the possible change events;
2. For each change event, perceive the alternative reactions.
3. Each path in the above decomposition graph is now an *evolution scenario*;
4. For each scenario, estimate the impact on schedule/quality/cost based on data in the baseline – the actual impact in the baseline project for similar scenario;
5. For each scenario, estimate its probability based on the frequency data in the baseline project;
6. If there are too many scenarios, select the ones with high probabilities and/or significant impact, and make decisions to change the project profile and/or the process model, or define appropriate contingencies to deal with the expected important changes.

Figure 9.5 shows examples of evolution scenarios and decision making for some typical evolution patterns.

9.4.4 Measurements and Risk Analysis

Systematic and valid process improvement can be achieved only by analyzing the process in a quantitative way, i.e. requiring measurements. For example the Goal-Question-Metric (GQM) method [BCR94b] that defines goals for software process improvement before embarking on software measurement tasks. Our method for predictive process evolution relies heavily on software metrics. Thus the project plan should include measurement of the product as well as of the process.

In our case, the process goals for measurements are:

- To monitor projects and their environments for factors that could fail, and to predict risks to schedule, budget, and product quality (risk analysis);

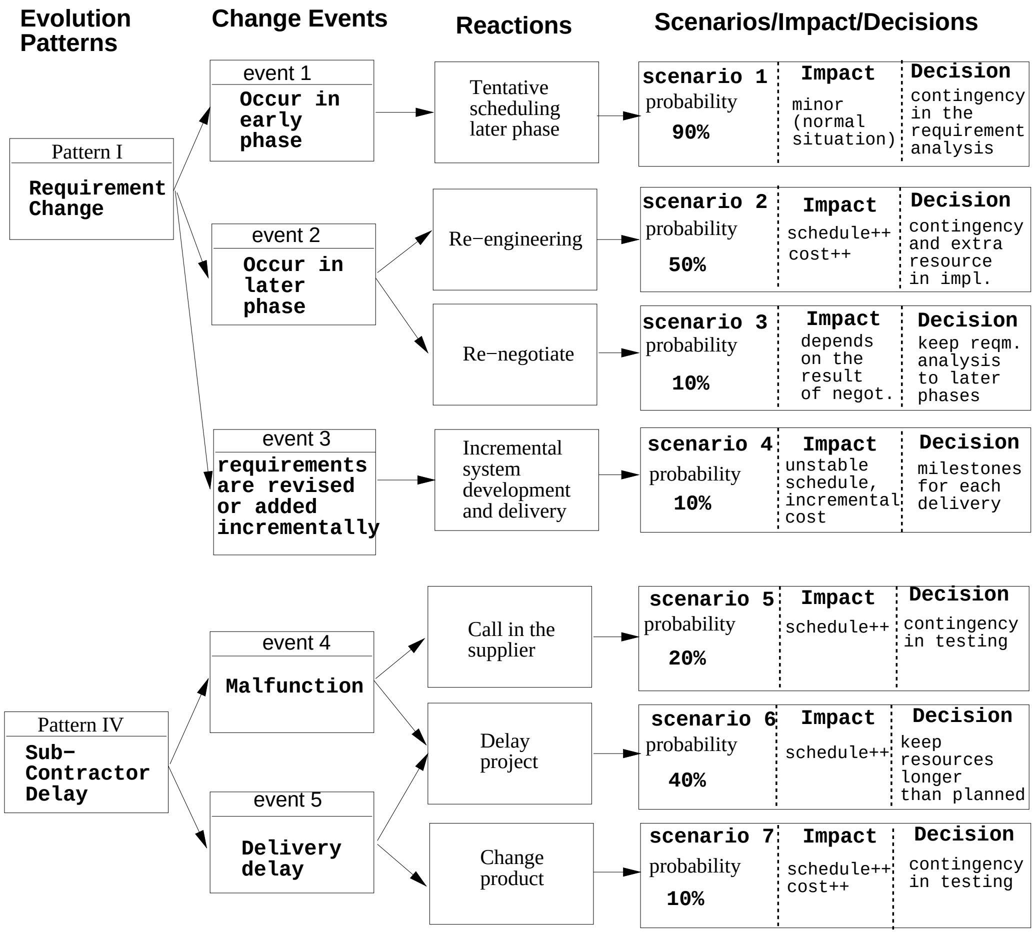

Evolution Patterns
Change Events
Reactions
Scenarios/Impact/Decisions
Pattern I
Requirement Change
event 1
Occur in early phase
event 2
Occur in later phase
event 3
requirements are revised or added incrementally
Tentative scheduling later phase
Re−engineering
Re−negotiate
Incremental system development and delivery
scenario 1 probability 90%
Impact minor (normal situation)
Decision contingency in the requirement analysis
scenario 2 probability 50%
Impact schedule++ cost++
Decision contingency and extra resource in impl.
scenario 3 probability 10%
Impact depends on the result of negot.
Decision keep reqm. analysis to later phases
scenario 4 probability 10%
Impact unstable schedule, incremental cost
Decision milestones for each delivery
Pattern IV
Sub−Contractor Delay
event 4
Malfunction
event 5
Delivery delay
Call in the supplier
Delay project
Change product
scenario 5 probability 20%
Impact schedule++
Decision contingency in testing
scenario 6 probability 40%
Impact schedule++
Decision keep resources longer than planned
scenario 7 probability 10%
Impact schedule++ cost++
Decision contingency in testing

Figure 9.5: Evolution Scenarios and Decision Making.

- To revise/refine the project plan to react to (anticipated) changes;
- To track change reactions for their cost and effectiveness;
- To gain experience for future projects; etc.

In the project plan, the following measurement mechanisms should be specified:

- The process/product attributes to measure during project execution;
- The data collection procedures, tools and timing;
- The mathematical/statistical/empirical models used in data analysis;
- The data analysis procedures and their results;
- How to use the results of data analysis, e.g plan revision, risk warning, future learning, and so on.

For a particular project, the project manager should focus on selected issues based on the expected evolution patterns. In the following, we give some examples.

1. Requirement Change – Evolution Pattern I

- *Data Collection and Analysis*:
  Count periodically the number of requirements (initially and number of added /deleted /modified requirements).
- *Risk Analysis*: If the requirements are still being modified, then the schedule is still at risk. If the total number of requirements is stabilizing, the risk is reduced.
- *Reactions*: If the risk to schedule is high, re-negotiate the budget or adopt the incremental system development and delivery approach.

2. Sub-Contractor Delay – Evolution Pattern IV

- *Data Collection and Analysis*:
  The estimated date D_1 of the availability of the external product *P* (based on its announced arrival time and the model for its acceptance testing) is periodically reported to management. In the project plan, the manager has the planned date D_2 for the availability of *P*.
- *Risk Analysis*: The risk to the project schedule and cost is inversely proportional to $D_2 - D_1$.
- *Reactions*: If the risk to schedule is high, consider replacing the external product *P*.

3. Internal Schedule Adjust – Evolution Pattern VIII

- *Data Collection and Analysis*: the effort data (person-hours) is collected by weekly reports to management showing recorded hours for each task. In the project plan, the manager has effort estimates for each phase and the total effort. The actual and estimated effort is periodically compared.
- *Risk Analysis*: Extensive deviation would indicate a potential problem area and risk to the schedule.
- *Reactions*: classical project follow-up.

4. Review-Modify Cycle – Evolution Pattern IX

- *Data Collection and Analysis*: Measure module characteristics (size, complexity, fan-in/fan-out, etc.). Use some model of correlation between module characteristics and module risk level. Applying the model to each module, we can estimate its risk level.
- *Risk Analysis*: To identify “high risk” modules that exceed their estimated or recommended complexity and size.
- *Reactions*: ”High risk” modules may need further investigation or even re-design.

5. Test-Modify Cycle – Evolution Pattern X

- *Data Collection and Analysis*: Estimate the total number of defects using industry guidelines, e.g. 1 defect per KLOC. Periodically count the defects found, and estimate the rate at which defects are found using some defect trending model. Then we can estimate how many defects that remain at any time of the project.
- *Risk Analysis*: If too many defects have not been found and the product is released shortly, then the reliability of the product may be unacceptably low and this is risky.
- *Reactions*: Prolong the period initially planned for the testing phase.

Based on different evolution patterns, the project plan should specify appropriate mechanisms for risk analysis and management (data collection, analysis method and models, risk analysis method and models, etc.) and pre-allocate necessary resources for reactions.

9.5 Preliminary Experiences

Parts of the (re)planning process in this paper has been prototyped, using the EPOS system [Ngu97]. Real projects were “simulated” in EPOS, with active process models. The empirical data were taken from a case study conducted with a Norwegian banking software house, called YYY [NWC97]. In that study we classified five historical projects and collected actual change occurrences in these. Organization YYY is ISO-9001 certified since January 1995, and has therefore a documented quality system. The purpose of the case study was to collect actual evolution data based on parts of the framework described in this paper. The preliminary analysis shows that most changes originate from customer delays or requirement changes. Many such changes occur in the design and testing phase, and constitute a major impact in term of extra cost. However, the case study and the initial EPOS implementation covers only part of the work presented in this paper. E.g., the classification system was not fully utilized, and replanning was largely incomplete.

The experience database for YYY has now been re-implemented in ORACLE, and is being populated with data from 30 past projects and tried out on 5 ongoing projects [Ing98]. The gained experiences is being fed back to the organization for making effective improvement decisions. YYY now uses a classification metrics called Project Implementation Profile (PIP) to specify the operating-profile [SS86], using 10 factors and with 10 values each.

The proposed classification and planning framework will be tried out on a larger scale in the Norwegian SPIQ project in 1997-2001, where SPIQ stands for “Software Process Improvement for better Quality”. Most of the 12 participating SPIQ companies including YYY are interested in establishing a corporate Experience Base to improve project estimation and generally to achieve SPI. Five of them are now building up small, web-based experience bases to improve their estimation capabilities. Their initial goal is better cost/time estimation of current projects, and not to manage unanticipated process changes. However, such process changes represent the (un)expected variation in such estimates, and are thus valuable information. In the next round, more up-front planning for process changes can be employed. For instance, one of these companies use five major tollgates in their projects, where they assess progress and (re)plan further activities. This is consistent with the proposed planning support in this paper.

9.6 Conclusion

The central topic of the paper is the planning support for software process evolution. This work presents not only research results in software engineering alone, but also transfers methods of knowledge engineering to software engineering, applying the AI planning technique to software process modeling and software project management. The main contributions in this work are:

- We propose a new classification of software projects, and several special *overall-profiles* of projects (time-critical, safety-critical, and prototyping) are identified apart from the classical time/budget constrained projects. In addition, several factors to constitute *operative-profiles* are introduced.

- We identify and characterize ten software *process evolution patterns* and link them to different project profiles.
- Based on the evolution patterns, we discuss the *planning support* for process evolution. The following methods are new or significantly extend existing work:
  - Cost estimation for *process changes*;
  - Evolution pattern analysis, and decision making for local contingencies or for more global SPI.
  - A total process for initial planning and iterative replanning, documented by a coarse process model (as pseudo-code).

The presented framework is a bit preliminary and needs detailing towards individual companies and cases. However, its core features have been applied with preliminary, encouraging results.

CHAPTER 10

Improving Cooperation Support in the EPOS CM System

Alf Inge Wang, Jens-Otto Larsen, Reidar Conradi, and **Bjørn P. Munch.**[1]

Abstract
This paper reports our experiences gained in designing, implementing, and experimenting with technologies for improved support for cooperative work in our configuration management (CM) system. The aim of the work has been to find a set of mechanisms supporting cooperation in a range of situations, from planning and scheduling long-lasting CM activities, to resolving access conflicts[2] between users. Although our tools are tailored for our home-grown environment, the general approach should be applicable also to other CM systems or usage domains. The emphasis of this paper is on flexible mechanisms to solve access conflicts without enforcing only one way of working.

10.1 Introduction

One of the problems in software configuration management regards cooperative work on large systems, where several people are doing development simultaneously, and their

[1]Norwegian University of Science and Technology (NTNU), N-7034 Trondheim, Norway, {alfw,jensotto,conradi,bjornmu}@idi.ntnu.no

[2]Access conflict does in this paper mean that two or more people want to change the same file at the same time.

work is not always independent.

A programmer may want to work in isolation, to avoid surprises when e.g. a library or a common header file is suddenly changed by someone else. On the other hand, (s)he may get in trouble when the work is finished, and turns out to be “wrong” because it is based on an outdated version of the same library/header file. Other kinds of problems arise when two developers want to change the same part of a product concurrently. To allow such an action, it should be allowed to have temporary inconsistency and support must be provided to ensure final consistency.

Situations like these inevitably show up in large development projects. Of course, it is always a good idea to try to organize the work in such a way that access conflicts do not happen too often. But it is not possible to avoid it totally without putting severe restrictions on the work. Instead, a CM system should give support for *planning*, *detecting* and *dealing* (this includes negotiation for solving conflicts of sharing data) with such conflicts, rather than trying to *avoid* or *work around* them at all costs.

To aid cooperation between users of a CM system, one should not only consider access restrictions to objects, but also provide awareness services to be able to see possible or actual access conflicts. Additional information can also be used to e.g. plan activities, negotiate about data sharing conflicts etc.

The EPOS CM system (ECM) is built on top of a general, versioned database (EPOSDB) and offers a set of commands to access software components stored in the database. We have added functionality in a number of areas to support cooperation among ECM users. In this paper we will go through the added functionality.

The rest of this paper is organized as follows: Section 10.2 discusses support for cooperative work within some existing CM and similar systems. Section 10.3 discusses about what cooperative support a CM-system should have and presents a list of cooperative support requirements for the EPOS-CM system. Section 10.4 is a short description of the EPOS CM system and what cooperative support we have added to the system. Section 10.5 describes the cooperative support we have added in EPOS. Section 10.6 presents some experiences we have had using the cooperative support in ECM. Section 10.7 discusses how cooperative CM support is related to the process modeling domain. The last section concludes this paper.

10.2 Related work

In this section we will give a brief presentation of some existing CM and similar systems, both commercial and research prototypes. We will describe their basic usage mode, mechanisms for concurrency control, and support for concurrent, cooperative work.

10.2.1 SCCS/RCS

SCCS [Roc75] and RCS [Tic85] are both versioning systems. Components are checked out from repositories and into file-based workspaces. To be able to return a changed component to the repository, the component must be locked, thus preventing other users from updating the component. Locks are released when components have been returned to the workspace. Neither system has a formal workspace concept, nor do they provide support for managing large workspaces. To summarize, these systems use simple component-based locking and do not provide any support for cooperative work apart from informing about who is holding locks on components.

10.2.2 ClearCase

ClearCase [Leb94] is a popular commercial CM system. Workspaces (Views) are central to ClearCase and provide a framework for specifying workspace-wide versioning behavior (version selection, branching, etc). Components are locked when checked out for update and the locks are later released. The additional support for cooperative work provided by ClearCase is support to locate, possibly access the changed components and support for handling software configurations on different distributed sites.

The cooperative support in ClearCase can be further improved by using ClearGuide. ClearGuide is a Software Process Management product – that try to guide software development teams through day-to-day activities, improving project coordination, and encouraging ongoing process improvement. This product is fully integrated with ClearCase.

10.2.3 NSE

NSE [Sun89, Fei91] (later TeamWare) has nested workspaces. Components are copied from the parent workspace, changed, and returned to the parent workspace. NSE does not lock components in parent workspaces, but has instead focused on detecting update conflicts and supports the user in merging own changes with concurrent changes from other workspaces when returning components to the parent workspace (during work and when closing a workspace). NSE also has a number of features to synchronize workspace contents with that of the parent. NSE allows users to go ahead with their work with maximum concurrency and provides good support for the process of integrating changes.

10.2.4 Adele

Adele [BEM93], has a number of high-level CM features and a powerful mechanism for automating workspace maintenance(triggers). Adele uses component locking and allows components to be exchanged between workspaces, even automatically. By using the trigger mechanism, Adele can provide some process support and also awareness support.

Apart from this functionality, upon which relatively advanced support can be built, there is no particular system support for cooperative work.

10.2.5 Lotus Notes

Lotus Notes[Orl92] provides a way of organizing documents and making them available to groups of people and individuals. The cooperative support in Lotus Notes is provided by the means of sharing information. This is done by using a document database where documents are stored. You can also use Notes as a GUI front-end for an information flow system or workflow system.

There is, however, no well defined configuration management support in Lotus Notes. It is possible to lock files manually and it is also possible to see latest changes to a text document as text shown in another color than the rest of the text. There are, however, no mechanisms to ensure consistency or to allow people to work in parallel.

10.2.6 Other Systems

Some dedicated Software Engineering DBMSes, like DAMOKLES [DGL86], provide open and flexible long transaction models, but lack a user framework that enables cooperation support. Software engineering environments like Marvel [Kai90b], COO [Cla93], and our own EPOS [MCJOLW95] all provide some cooperation functionality based on their own databases.

More traditional groupware systems like TeamRoom [RG96], BSCW [BHT97] and Object Lens [MLG92] provide support for awareness and group-sharing, but little or no support for versioning management. These system provide workspaces with only limited or no support for locking of documents and for dealing with different configurations of these. However these systems support negotiation between users of the system.

10.3 Cooperation requirements

Software engineering involves large data sets at client sites and long update times. Since the scope and sequence of updates are hard to predict and may involve overlapping/ versioned subsystems, traditional locking procedures may cause intolerable delays. Thus, software engineering – like concurrent engineering in CAD/CAM and VLSI – also requires support for long-lasting and user-controlled transactions, often called *design transactions* [KKB85], or workspaces in the CM context.

From the discussion in section 10.2, we see that the most common way to handle concurrent work by many users is to avoid update conflicts by locking components (only NSE uses another approach). Users will then have to communicate to decide how to handle the access conflict: creating temporary versions to allow concurrent work, waiting for the other user to finish work, or to work around the system (at the cost of loosing system support).

Design transactions are not, however, enough to solve all problems regarding concurrent work on the same files in a CM system. There must also be other mechanisms that can help users at a higher level to do the right decisions before and during conflicting situations. Some of these mechanisms are functionality typically provided in groupware systems, some are related to scheduling software, while other mechanisms are related to advanced CM systems.

Soft locks can for instance be used indicate that someone is changing a specific file. In contrast to traditional locking, soft locks do not ensure consistency. This can, however, be done through merging and negotiation.

If traditional locking is not enforced in a system, there are many ways to help users to handle access conflicts that will occur. To make sure that access conflicts don't occur, it is possible to plan the file access in advance. If no such actions has taken place, the system can make users aware of what others do and what objects they access. To resolve access conflicts when they arise, it is possible to use negotiate procedures and to exchange products between workspaces to merge changes. Some properties essential to a cooperative CM system should be:

1. *Shared plans*: Planned and ongoing activities, both small fixes and larger efforts, should be described and entered into the system so that other activities can be planned as scheduled based on this information.

2. *Workspace information*: By making workspace related run-time information available in an easily accessible format, users are able to reason about the causes of conflicts.

3. *Awareness*: Providing support for notifying users about events that will affect their work in their respective workspaces.

4. *Communication infrastructure*: A cooperative system should include a system for sending messages and notifications, both user and system generated information. An integrated infrastructure will improve communication precision and performance.

5. *Flexible locking mechanisms*[3]: To manage concurrency problems, different lock modes allow more concurrent work in a system-supported manner. The underlying point is that with a formal criterion for correctness of concurrent component access, we are able to reason about the inconsistencies that may arise, if we choose a conflict-detecting instead of conflict-avoiding way of working.

[3]*Flexible locking mechanism does in this paper represent a extended locking mechanism compared to traditional locks that provides different lock-types (read locks, soft locks etc).*

6. *Component exchange*: There should be a way to exchange a copy of a component between workspaces, before checking it back into the repository (pre-commit exchange).

The next section describes how our CM system has been extended to support these requirements with emphasis shared plans(1).

10.4 ECM - the EPOS CM system

This section gives a short introduction to the EPOS configuration management system (ECM) and introduces the cooperative support that has been added to the system. For more detailed description on ECM look in [LMCL95].

Figure 10.1 shows the main parts of the EPOS CM system with the cooperative support added. The main parts of the system is the ECM tool and the EPOSDB [Mun93]. The ECM tool has both a graphical and command-line user interface and provide support for:

- **Manage workspaces**: Navigate in workspace hierarchy[4] and create new/abort/commit workspaces.
- **Manage components**[5]: Navigate in the product space, check in and out components etc.

The EPOS Database [Mun93] is a general-purpose DBMS with features geared for CM. All objects are versioned according to our CoV versioning model [MLG^{+}93], which we will not discuss in this paper.

The shaded blocks in figure 10.1 represents the cooperative support extensions in EPOS. Here is a short description of the extensions:

- **ECM Tool Extra cooperative support**: Send user messages, conflict detection, synchronize file support, browse workspace information and merge file support.
- **Awareness services**: Browse/Subscribe notifications and automatic update of newer versions of files.
- **EPOS Message Server**: Distribute system messages.
- **Component exchange**: Exchange components between workspaces before the workspaces are committed.
- **EPOSDB schema extension**: To support Workspace information, flexible locking and Shared working plans

[4]EPOS supports nested workspaces which is represented in the repository as nested transactions
[5]The term component in EPOS is used to name products, files etc.

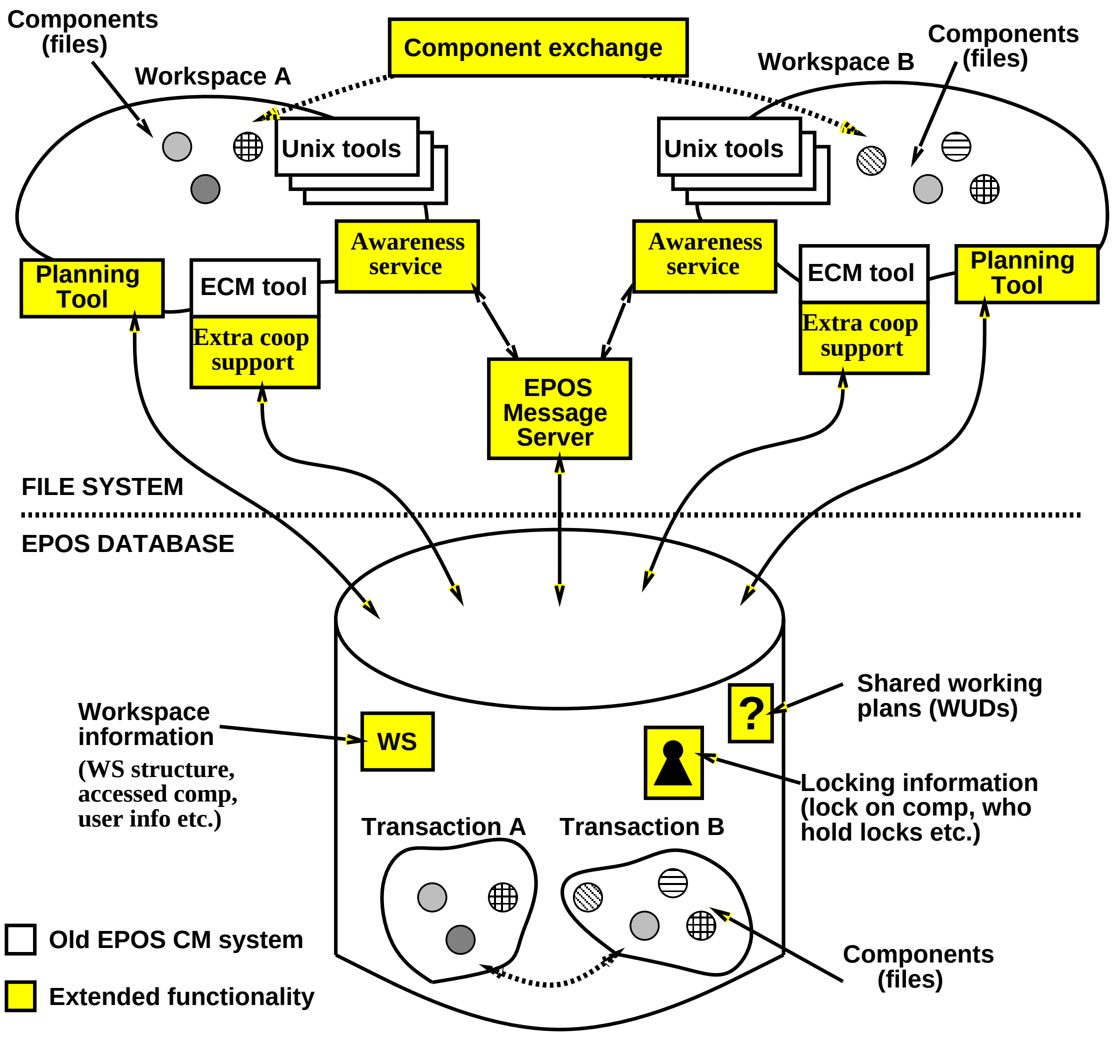


Figure 10.1: EPOS CM with cooperative support added.

- **Planning tool**:Plan file access in advance to avoid conflicts.

More detailed description of these features is shown in section 10.5

10.5 Extended cooperative support in EPOS

This sections describes the the cooperative support we have added to the ECM system. The cooperative support described in this section is not only restricted for EPOS and could be applied to other CM-systems as well.

10.5.1 Shared plans: Work-Unit Descriptions

We want to provide support for declaration of work intentions. This means planning information (e.g., resources, calendar and dates, and expected work sets), are specified in a formalism comprehensible to both users and the system. This permits us to reason on the plans against the actual state of a project: potential cooperation problems can be identified and solved in advance. These specifications are primarily meant for a priori coordination of work.

The term work-unit (WU) covers the notion of a "meaningful" unit of change/ revision made to software objects in the EPOSDB. By storing work-units in the EPOSDB, it is possible to share descriptions of planned work with other users, so that it will be possible to coordinate and schedule larger change jobs to minimize the potential for access conflicts.

The intentions of the work to be performed in a work-unit is given by a work-unit description, WUD. This description primarily encompasses information needed to start an ECM workspace, such as version configuration[6] and read-/write-sets of named software components. WUDs are stored in the EPOSDB using a schema which is structurally similar to the one shown in figure 10.2.

WUDL ("Work Unit Description Language") is a textual language for defining properties of and pre-declaring work intentions for a workspace or transaction. WUDL specifications are declarative, and their purpose is to define in advance which objects will or may be accessed, not when and how. The most important information found in a Work Unit Description is *Read sets*, defining what files will be read, and *write sets*, defining what files will be changed.

We have implemented a planning tool [CLH95], as shown in figure 10.1, which can analyze a set of work-unit descriptions and suggest an execution history which will minimize the set of dependencies between concurrently executing transactions.

[6]The version configuration is in EPOS represented through ambition and choice as described in [Mun93].

WUDL can be extended to contain elements for coupling ECM closer to the EPOS software process support tools. A longer-term goal would be to supply a library of concurrency control and cooperation policies which cover the most methods found in database systems. Using this library, we are able to specify work-units at a relatively high level of abstraction.

10.5.2 System information: Workspace Information

Workspace information is stored in the database and maintained when users are performing operations related to workspaces. Some examples of the information are: A general description of the work intention, the workspace structure (hierarchy), and the set of accessed objects along with lock modes. This information is accessible through a set of tools/commands, and can be used to detect and/or avoid access conflicts at any time.

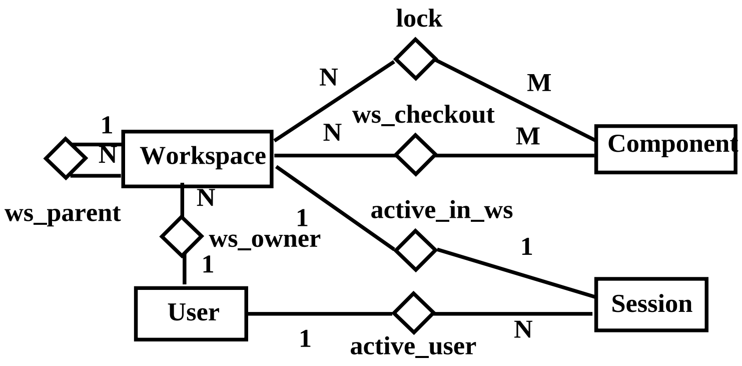


Figure 10.2: Schema for transaction and workspace database.

To support interactive conflict solving, we need to supply sufficient information for the users to be able to locate the source of a conflict.

Our solution is a meta-database of workspace-related data. This database can be queried from within any workspace and it is updated by various access and transaction control operations. The transaction and workspace meta-database will give a consistent view of which transactions exist and which objects are locked or checked out by which transactions.

The main parts of the ER-schema for this meta-database is shown in figure 10.2. The workspaces are modeled by an entity type and the hierarchy by the `ws_parent` relation. The information stored in workspace entities is basically what is described in the WUDL section 10.5.1. The `lock` relation type models locked components, and similarly the `ws_chkout` relations show which workspaces have checked out a component. The `user` entity type contains information about a user of the CM system (name, email, office, phone number) and which workspaces are owned by the user. If the user is active, there will be a `session` entity for each ECM client running, containing host and process information.

ECM has graphical browsers which display this information (referred in figure 10.1 as ECM Tool Extra), and the user can also query this part of the repository through the command-line interface using the same commands as used for other database queries.

The goal has been to build a system where one can easily find the source of a conflict arising from an attempted operation. The basic scenario is that a check-out operation fails because the component is already checked out in another workspace. The user can then find which workspaces have checked out the component, and which is causing the lock conflict. From the workspace entity one can find the user owning the workspace and, if any, the active users. The user will then have to take appropriate actions to resolve the situation: delay the operation, undo one's own work, or request the partner involved in the conflict to act.

10.5.3 Awareness

Whenever a user invokes a command that will update the database contents, a *notification message* is sent to all users that subscribes to this type of message and that may be affected by the event. The notifications are handled by the EPOS Message Server as described in section 10.5.4.

The notification messages are displayed and managed through a separate user interface, with features for filtering out the notifications that are not interesting for the user. Notification about an actual access conflict cannot be filtered out. The two most important message classes are:

- **Possible write/write conflict:** This message is used to tell a user that other users have checked out the same component for updating. This makes it possible for a user to abort the check-out of a component for updating to avoid an access conflict. The message will carry information about what workspaces the conflicting components are checked out to and what users are currently connected to these workspaces. EPOS CM has tool support for performing a negotiation process which can be utilized for solving access conflicts.

- **Actual write/write conflict:** This message is sent if two or more workspaces have checked out the same component for updating and checked in the components into the workspace. This situation means that if the conflicting workspaces commit, an actual write/write conflict occur.

  At this stage it is possible to solve the access conflict in at least three ways. The first and the least wanted way is to abort one of the workspaces (changes will be lost) The second way is to synchronize the changes to a component made by two or more workspaces. EPOS CM has a tool for going through the steps in a synchronization process. The last way of solving the access conflict is to do nothing before committing the workspace. The ECM Tool will automatically detect the components that are in conflict with other workspaces. A merge tool will be invoked which suggests how the files should be merged.

Another awareness service that EPOS CM provides is automatic update of a component. This means that when the system detects that there exists a newer version of one component, you have checked out into your workspace, there is support for updating this component.

More detailed information about the awareness support in EPOS is described in [Wan95].

10.5.4 Communication infrastructure: ECM Message Services

To enable asynchronous cooperation between software developers using ECM, we have designed and implemented a messaging system which is used for notifying users about events, e.g. database access, updates and commit operations, originating from other workspaces. Events are specified in terms of ECM operations, so that users can easily relate to the notifications.

ECM Tool provides a user interface for displaying and browsing messages, and replying to them or sending user written messages. The messages are stored as entities locally to the receiving workspace.

This message system is primarily useful for giving early notification of actual or potential conflicts in the case that workspaces use non-restrictive locking or no locking at all. A workspace can subscribe to events and will receive a notification message whenever the specified event takes place. It is possible to subscribe to specific types of messages (related to specific events) and it is also possible to subscribe to events related to specific components or specific workspaces.

The EMS (“EPOS Message Server”) is responsible for sending messages between workspaces. ECM commands like check-in and check-out trigger the EMS, which will then send the notification messages to all workspaces subscribing to the event.

A possible extension to EMS is a system which can respond to EMS messages and perform specified actions, e.g. automatically check out a component which has been changed in a sibling workspace.

The awareness services described in previous section makes use of the ECM Message services.

10.5.5 Flexible locking mechanisms

The flexible locking mechanism in EPOS is provided on two different levels; the database level and the workspace level. The following will describe support on both these levels.

The EPOS Database locking support

In essence, the EPOSDB provides large set of lock-modes that can be used for enforcing a variety of correctness criteria with different implementations. The responsibility for guaranteeing correctness is partly left to the user and/or applications, in our case the ECM Tool or ECM users. However, there is a default behavior that provides failure atomicity and partial isolation.

Workspace locking support

When components are checked out into the workspace, they are locked in the repository to control access by other users. We have defined 7 lock-modes which can be used to achieve a range of access control behavior, spanning from the very restrictive to giving warnings. Locks on components can be set through the Extra support in the ECM Tool (see figure 10.1).

10.5.6 Component exchange

The basic workspace model (and the corresponding database transaction model) has been extended with operations allowing *flexible and system-supported exchange of objects* between workspaces. This feature provides support to synchronize workspaces, further to support merging changes to keep components consistent and also if user want to interchange components between workspaces.

In our CM system this is reflected as commands to move or copy components between workspaces (through the database), both between ancestor and sibling workspaces. These commands are used to exchange components between users/workspaces in a controlled manner and at a fine granularity.

10.6 Experiences

This section presents some experiences we have so far from using cooperative support in ECM.

The general impression from experimenting with cooperative support in ECM is that allthough we have a flexible system this has its price. The philosophy supported in ECM of supporting different working modes means also that if one person wants to use only a minimum set of the cooperative functionality, the rest of the users might suffer from this. For instance, if one user don't want to create any shared plans, the rest of the users will not be sure if they will have access conflicts or not. Another example is if one person does not want to merge changes to a component with another persons changes, this would lead to a fight. From these experiences we found it was necessary to have a set of cooperation protocols that should be followed by people sharing the same components. We experienced that if people did not agree on how to share the components before they started to work, the users would not benefit from the cooperative support in the system. However, if people agreed to follow a cooperative protocol, the system would provide cooperative support.

We have identified four overall cooperation protocols that was useful to support sharing of files:

1. **Commit-Merge**: People work individually without caring about other people until the workspace is committed. Conflicting components must then be merged.

2. **Merge-Commit**: People synchronize components in access conflicts before the workspace is committed.

3. **Exclusive write lock**: Components that are updated are locked with a write lock as in SCCS/RCS.

4. **Exclusive lock**: Components that are updated are locked with a write as well as a read lock. This means that no one can read changes to components before the workspace is committed.

For the each of the four cooperation protocols above, different cooperative support services can be applied. The Commit-Merge protocol imply that people don't care about cooperating while changing the components. When merging the components, there might be required to have support for negotiation between involved partners. The Merge-Commit protocol, the emphasis is on awareness support. One need to know of access conflicts when they occur, so a synchronization process can start between the involved workspaces. Negotiation support is also needed for this protocol.

The two last protocols we have listed emphasis their cooperative support on shared plans. In this scenario it is important to avoid people working on the same components at the same time. Shared plans can then be used to create a plan for who can change components when etc. Shared plans can also be used to find the reasons for locking components.

From the experiences described above, it seams like some cooperative protocols demands a strict process to be followed, while other cooperative processes are more ad-hoc. Next section will go into this more in detail.

10.7 Discussion

This sections discusses how the cooperative support in CM-systems relate the process modeling domain.

All though EPOS also has a variety of tools to model and execute process models (EPOS-PM), we did not use EPOS-PM support to implement cooperative support in ECM. There are at least three reasons for this.

First, cooperative processes are often hard to model because they consist of many interactions between involving actors. Most Process Modeling Languages (PMLs) focus on activities and relationships between them, while for cooperative processes the actors and interactions between actors are in focus. Clearly, cooperative processes is out of the domain for most PMLs and thus can not be modeled in these languages.

Second, cooperative support must add little overhead work to the involved actors. Most PM-systems requires a lot of work to get the processes up running. Since cooperative processes often vary in the way they are, you must in most cases have to create a new model every time you need some cooperative process support. One of the main requirements for processes that could benefit from having process modeling support is that the process is repeatable. Since this is seldom the case for cooperative processes, PM-support would be just waist of time.

Third, cooperative processes evolve all the time. Even if in some cases there will be cooperative processes that are similar, these processes would in most cases have small changes or exceptions that would be different from time to time. This means that the PM-system need to be able to support changes on the fly. All though some PM-systems (including EPOS-PM) provide flexible support for handling changes to the process on the fly, the operations to do these changes will offer to much time. Thus the user would not benefit from the PM-system.

From the discussion above it seams like PM-systems can not provide any cooperative support to a CM system. This is not always the case. The ”Shared plans”, as described in section 10.3 and 10.5, often require a strict process to be followed if access conflicts should be avoided. This means that the shared plans could be used to model the process and the PM-system would provide support for executing the process. The Workspace Information as described in section 10.5.2 might also provide useful input for a PM-system. The PM-system can for instance benefit from information about who is working with what and what files has been changed etc. A important question to ask here is if the PM-system should be allowed to access these sensitive data that would make it possible to for example to measure progress of people.

10.8 Conclusions and further work

We have presented a CM system which has been extended with mechanisms to support cooperation between users of the system. We have discussed different types of information that can be entered into the system and how it can be used for planning, conflict detection and identification of partners involved in both possible and actual conflicts. We have presented tools that can be used to retrieve information on demand and a tool that can display messages about events caused by other users.

By integrating the cooperation support with the CM system, we are able to offer more precise information related to the objects managed by the CM system, and result in more reliable and easily maintainable information.

The users also have support from the CM system for propagating changes to each other, so that they can resolve the conflicts resulting from parallel work. The system is customizable and can be used to support a range of work modes, from loose to tight cooperation.

The idea of integrating work-related information into the CM system should be applicable to other systems, even much simpler ones. We would need to implement tools to enter and retrieve this information, and users must be made aware of the importance of providing information about work intentions.

This paper presented how cooperative support can be provided in a CM-system. We have also look at the relationship between cooperative support in a CM-system and process modeling support. It seams that there is a gap between the CM- and PM-system in this respect and future research should try to minimize this gap.

CHAPTER 11

Teaching Software Process Improvement through a Case Study

Torgeir Dingsøyr, M. Letizia Jaccheri, and **Alf Inge Wang**[1]

Abstract

This paper describes the main design choices of a software process improvement course. The course is organised around an industrial case study. In addition it is based on lectures and group exercises. The case study is centred around four research questions: Why is process improvement important [in your company]? Which processes does your company have? Which improvement initiatives does your company implement? Which relationships exist between software improvement and software quality?

During the case study, the students come in contact with actors from the local software industry. We experienced problems with with relating quality and process issues, and that the insight was too superficial. We also had problems with student involvement.

Finally we propose a new set of questions that are: Briefly describe a software system that is (or has been) important for your company. Which attributes do you use to describe it? Which are the processes around this system? Which are the improvement initiatives around these processes? How general is that specific software system (and respectively

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7035 Trondheim, Norway. Phone: +47 73593444, Fax: + 73 594466, Email dingsoyr/letizia/alfw@idi.ntnu.no

its processes and improvement initiatives) in the context of your company?

11.1 Introduction

Software quality and software process improvement are central topics in modern IT industry. However, there is no standard consensus about how to educate future software engineers in such topics. Software quality and software process improvement comprise both technical and managerial issues. Among the technical, we list design, testing, inspection, and configuration management. On the other hand, the quality and improvement models, like Capability Maturity Model [PCCW93], ISO 9000 [KJ95], Quality Improvement Paradigm [BG94], etc. derive from managerial and organisational theories. When teaching technical methods, such as design, one can employ the same educational methods that are commonly used to teach programming languages or mathematics: First, the teacher explains the method and provides examples. Then, the students are asked to solve toy problems, alone or in groups, by employing the method. This educational method does not properly work when applied to teaching of the managerial part of software process improvement and software quality.

Since 1988, the Department of Computer and Information Science has been responsible for teaching a *Software quality and process improvement course* (SQPI course) for 4^{th} year students at the Norwegian University of Science and Technology (NTNU). In the beginning, the course concentrated on technical topics, such as design, programming, testing, inspection, and configuration management. The educational method were the classical one. Over the years, the contents of the course has changed to gradually to include a survey of software quality and improvement models, such as CMM and ISO9000. However, the teaching methods have been stable.

The teacher presented motivating examples and figures. Typical examples include the amount of money lost on failed software projects the last 20 years. However, many of the examples were too distant from the students as they mainly concentrate on North American IT industry. The course did not include practical exercises on the managerial part of quality and improvement issues.

Some of the 4^{th} years students have worked in Norwegian software companies and are familiar with problems caused by failed software projects. Also, other software engineering courses rely on student projects with industry actors playing the customer role. In spite of all, it was hard to motivate students with examples that are far away from companies they have worked for or will work for in the future.

The result was that the course failed to get student involvement. Students got a surface knowledge of the managerial topics, but were not able to relate them to the technical ones. In addition, they were not able to relate the improvement and quality topics to the local Norwegian IT reality.

In 1997, we redesigned the *SQPI course* with the goal of making students get a deeper understanding. We wanted the students to understand how different subjects in the course were related, and how the course relates to the local IT industry. In this paper we will discuss the course main design choices. The *SQPI* course is organised around a case study where students interact with actors from the IT industry by interviewing them by means of a set of research questions. The students are asked to write reports about their perception of actor answers.

The rest of this paper is organised in three main parts. Section 11.2 describes the main choices underlying the *SQPI course* and provides an evaluation of the course. Section 11.3 looks at similar related work, identifies the problem we detected with our approach and suggest how we should remove this problems in the future.

11.2 The Software Quality and Software Process Improvement course

In this section we describe the main design choices taken when preparing the course, and also give an evaluation of the course.

11.2.1 Course Design

This (sub)section describes the plan of the course Software Quality and Software Process Improvement for year 1998. Next section 11.2.2 will discuss course implementation. The course design for year 1999 will be sketched in section 11.3.2.

The course is based on the following elements and their interaction:

1. A theory part. This is based on:

   (a) The book ”Managing the Software Process” [Hum89]. We chose this book as it is a solid book used by a lot of software engineering courses that address software process issues. Other alternatives are [Gra92], [Gra97], [Cap98], [PWCC95], [Fen91], [Zah98], [FKN94] and [Som95]. The big disadvantage of our choice is that the book concentrates on North American software industry and little on European.

   The students are supposed to read the book early in the semester, in parallel with lectures, before the case study begins. The students are encouraged to find discrepancies, inconsistencies, and common points between book, lectures, and case studies.

   (b) Lectures and related material handed out by the teacher. Here, we introduce the concepts of software development and maintenance process, software quality, and software process improvement.

The teacher distributes the slides before each lecture. Slides are also published on the web-page of the course [JW99]. There are 10 hours of lectures, see entry labelled by teacher in table 11.1.

Day	Type	Responsible	Topic
Fri 23/1	Lecture	Teacher	Course Introduction
Mon 26/1	Lecture	Teaching Assistant	Introduction to Exercises
Fri 30/1	Lecture	Teacher	Sw Process
Mon 2/2	Lecture	Teacher	Sw Quality
Fri 6/2	Lecture	Teacher	Sw Process Improvement
Mon 9/2	Industry Presentation	Telenor Novit	Case Study
Mon 9/2	Deadline	Students	Inspection Exercise Delivery
Fri 13/2	Deadline and Presentation	Students	Delivery and presentation Telenor Novit
Fri 20/2	Industry Presentation	Statoil	Case Study
Fri 27/2	Deadline and Presentation	Students	Delivery and Presentation Statoil
Fri 6/3	Industry Presentation	Ericsson	Case Study
Fri 13/3	Deadline and Presentation	Students	Delivery and presentation Ericsson
Mon 16/3	Lecture	Teacher	Discussion and Conclusions
Mon 27/3	Deadline	Students	CM Exercise delivery
Mon 27/3	Lecture	Teaching Assistant	Information about Exercise Process Modelling
Mon 17/4	Deadline	Students	PM Exercise Delivery

Table 11.1: Semester Plan, each lecture corresponds to two hours.

(c) Group exercises related to theory. Groups consist of four students. There are three exercises. The first consists of inspection of a piece of code (742 C++ lines). The second is testing and configuration management of the same piece of code which is used in the first exercise. The inspection and testing techniques to be employed are those explained in the text book. The third exercise consist of modelling of a part of a software process by means of the E^3 [M.L98] formal process modelling language. We have chosen these exercises as the text book puts emphasis on testing, inspection, and defined process as first steps toward software process improvement.

Exercises are mandatory. Students cannot take at the exam without having delivered their exercises. The evaluation of exercises do not influence the exam grade. All exercises are delivered both on paper and electronically. This is valid also for the group exercises related to case study.

2. A case study.

This consists of:

(a) Three presentations by actors from three Norwegian software companies. Each actor is asked to give an answer to 4 questions. The questions are:

i. Why is process improvement important [in your company]?

ii. Which processes does your company have?

iii. Which improvement initiatives does your company implement?

   iv. Which relationships exist between software improvement and software quality?

(b) Group exercises related to case study.

   Each group is asked to write a document and to prepare a presentation that describes the actual presentation, how and if the questions are answered, which are the relationships with other presentations, the text book, and lecture.

(c) Presentations by students and related discussion.

   Each time, the teacher chooses 2 groups randomly among the total 10. Each group has 30 minutes to present its contribution. 15 minutes are devoted to discussion.

The exam is written and it has a duration of four hours. It is an open book exam. Each student is asked to answer four questions:

1. Case study. The student is asked to evaluate one of the four research questions and give examples. This one counts 30% of the final mark.

2. Process modelling related material. The exercise is to model a process fragment presented by one of the three industry actor by mean of the E^3 [M.L98] notation. This counts 35%.

   This question is intended to test how each student has assimilated stuff related to exercises. Process modelling was chosen as it was possible to relate it with a process that was distributed by an industry actor during the case study.

3. Book and material distributed by the teacher. The exercise asks to apply the function point formula to the piece of code used during inspection and CM. The code is provided. This one counts 15% of final mark.

   This counts only 15% of final mark as the material is available to the students during the exam.

4. General comprehension of the whole course. It asks about the relations between Configuration Management and software process improvement. It asks for examples from case study and other experience. This counts 20%.

11.2.2 Evaluation

Our main requirements have been fulfilled: students met at lectures and participated actively to the course implementation. We were able to provide real examples from the Norwegian software industry. This section discusses the main results of the course including exam results.

Table 11.2 shows how the *SPQI course* scored in average on the students evaluation of the course. The evaluation is a standard form used at NTNU consisting of nine questions. In questions 1-8, the students must choose from a scale from one to nine to indicate if they are positive or negative to the question. Score one is a very negative answer, while score nine is a very positive one.

No.	Question asked	Mean
1	Was the goal for the course made explicit ?	6,9
2	Did the teaching stimulate you to work with the course ?	6,4
3	Did the teaching of the course fit previous knowledge ?	7,4
4	How do you judge your own work with the course at this time ?	5,6
5	Does the load fit with the load given in the course description ?	5,7
6	Does this course build on thing you've learned in other courses ?	6,4
7	How is the teachers presentation technique ?	7,3
8	Does the exercise teaching work well ?	6,4
	Mean value of all questions	6,51
	Standard Deviation of all questions	0.66

Table 11.2: Students evaluation of the Software quality and process improvement course 1998

.

Table 11.2 indicate that students are above average happy with the course. Questions 1, 3 and 7 have scored high on this evaluation. This indicate that the students were very pleased with the presentation of the goal of the course, that the course fit to their prior knowledge and that the teaching technique of the was very good.

To make a more detailed description of the course evaluation, we want to present some of the more detailed written evaluations of the course:

- "It's very good to have companies to get their view of the course' subjects. It is also interesting to see actually how they work in practise (not only theory). The book is ok, but maybe a bit boring (too many details). I am very happy I chose this course."
- "The teacher did not lecture the book directly: Good. Maybe the teacher should have chosen to follow the book more anyway."
- "The book together with the exercises complements very well and give look at various subjects from different views. This inspires to be critical to the subjects taught and to think on your own. The book is maybe a bit too americanised."
- "The book is very good and points out many interesting things, but is maybe a bit ambitious in some areas. I think this course has been one of the most interesting course I have been taking. One thing can be changed: Too much work is spent on exercises in this course. This means I don't get enough time to do other courses."

- "The course is a bit different with company presentations, but this force us read through the book and relate this to present practises. The workload on the exercises is to high. I didn't like the student presentations either (it didn't work well)."

Most of the students seemed to like the changes we have made. There were some critics about the workload in the course, specially on the exercises. Many students wrote that they were motivated to continue read more about subjects presented in the course.

Exam results

Table 11.3 shows the mean value of the exam results the last five years as well as the standard deviation of the results. Grades are given from 1-6, where 1 is best and 6 worst. In order to pass a course the student need at least the grade 4. Grades are given in 0.5 intervals.

Year of exam	No. of students	Mean	Standard Deviation
1994	36	2.58	0.76
1995	48	2.34	0.59
1996	50	2.31	0.57
1997	34	2.57	0.99
1998	43	2.24	0.44
Total mean	42.2	2.41	0.67

Table 11.3: The exam results the last five years

The table shows that the year 1998 has the best mean of grades of all the five years and also that the standard deviation of the exams grades are lowest. The following can explain this:

1. Fewer and more general questions make it easier for students to write sensible answers on the exam (harder to write completely wrong answers).
2. With more student involvement in the course, most students actually got interest in the course and were motivated to work enough with the course to achieve a good grade.

Looking more closely at the individual results from the exams the last five years, we also found that in the year 1998 none achieved the best grade (1.0), but none failed the course. One of the main changes in 1998 compared with earlier years, was how students participated in the course. It was required that the students played an active role themselves by asking and commenting on how theory and practice relates. Most of the students had to make presentations that compared these things, which forced them to read the book and understand the various subjects in the course.

11.3 Conclusive remarks

The section first looks at how our work relates to other works, the results of our work and where we would like to go from here.

11.3.1 Related work

A general approach to teaching software engineering is to relate the course to either industry or a simulated environment (see [JL98] for a survey of project oriented work). Other courses at our department at the Norwegian University of Science and Technology have made extensive use of project-based work that involve both students and industrial actors [ACK$^+$94].

As discussed in [USK97], software engineering process education sets extra requirements. The main requirement seems that of providing the students with visible processes. This would mean to let the students work at the management level.

Our approach differs from these ones surveyed in [JL98] as our students do not work in a product oriented project, but they rather observe quality and project initiatives in industry.

Georgia Institute of Technology runs a course called Real World Lab [MB95], where undergraduate students are involved with real industry projects with products and customers. In addition, students take part in performing a CMM assessment on local industry by interviews. The difference from our approach is that we use a normal auditorium setting rather than project work. This is due to the project workload in other courses, and also to the fact that there are few companies in Norway that can illustrate a CMM level above one.

11.3.2 Summary

We have designed a software process improvement course around the integration of a case study and a set of lectures and exercises. The case study is centred around four research questions. Students must elaborate and present their perception of process and quality initiatives in the companies.

Our main requirements have been fulfilled: students met at lectures and participated actively to the course implementation. We were able to provide real examples from the Norwegian software industry.

Howevere there are some problems.

- **Superficiality** The work was generally too superficial. Participants from industry and students did not have enough time to address all the issues that had been planned. Each actor had two hours to give a whole picture of software process improvement and quality issues in his/her firm. And the picture was too general. Also the students had to produce three documents, and each document had to address the four questions.

- **Quality versus process** Software quality issues were not adressed enought. While the three company presentations adressed software process improvement issues, they neglected the software quality ones. Nobody provided examples of real software products. Really, the fourth question "*Relationships between software improvement and software quality* was not answered. This may come from the fact that participants were people working at the quality manual level and not system developers.
- **Student participation** There was not enought interest in student presentations by those students who had already presented.

11.3.3 Further Work

To obviate these problems, we have designed a new plan for the course that will be followed during the Spring semester 1999. According to the new plan, the theory part and the group exercises related to theory.will be the same as last year (see section 11.2.1). Concerning the case study we plan the following main changes:

There will be a couple of actor from **one Norwegian software company** who will represent both the process and the product view of the organisation. This should solve the *superficiality* problem.

To solve the *Quality versus process* problem, the questions asked to the software companies will be the follwing:

1. Briefly describe a software system that is (or has been) important for your company.
2. Which attributes do you use to describe it?
3. Which are the processes around this system?
4. Which are the improvement initiatives around these processes?
5. How general is that specific software system (and respectively its processes and improvement initiatives) in the context of your company?

Provided that there will be circa ten student groups as last year, we will have each group working a given answer in a way that for each answer there will be two groups working on it. In this way the quality of work around each question will hopefully be higher. Also, since students are required to be acquainted with the whole set of questions, we hope that there will be more interest in other group presentation. This would evantually solve the *Student participation* problem.

CHAPTER 12

Experience paper: Using XML to implement a workflow tool

Alf Inge Wang[1]

Abstract
This paper presents experiences we had from building a workflow tool from scratch using XML technology. We will present some strengths found using XML-technology, but also some weaknesses. Although we had to create a simple process modelling language for this workflow tool, the focus of this paper is on experiences on using XML technology to build workflow tools. The experiences we have achieved, should be applicable for all kinds for process modelling languages. The paper consists of three main parts. First, the requirements for the workflow tool is outlined. Then XML technology is explained with some simple examples. The last part of the paper describes experiences we achieved from the experiment and the conclusions we drew from this.

Keywords: *Process modelling language representation, workflow tools, XML*

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), 7035 Trondheim, Norway. Phone: +4773594485, Fax: +4773594466, Email: alfw@idi.ntnu.no

12.1 Introduction

Spring 1998, the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU) was asked by a project called Renaissance, to create a simple workflow tool to demonstrate the *Renaissance process*. The Renaissance project was a partially founded project by the European Commission under the Framework Initiative (ESPRIT 22010). The main objective of the Renaissance project was to develop a systematic method to support the re-engineering of legacy systems. Among the results of the Renaissance project was the Renaissance method described in the Renaissance method book [Con98]. This book describes a step-by-step process for re-engineering legacy systems in an informal graphical process language. Our assignment was to create a simple graphical web-based workflow tool that made it possible to go through the whole process in an interactive manner.

Our research interest for this assignment was not to create yet another Process Modelling Language (PML), but rather to see what technology to use to build a simple workflow tool over a short period of time and with scarce resources. The PML we chose, represents a process as a activity network interconnected with artifacts. Activities are activated through pre-conditions constrained by the states of their input artifacts. Although this papers will outline how the PML is represented in XML, the focus of this paper is on experience using XML to build a workflow tool (not the PML itself).

The rest of this paper is organised as following. Section 12.2 outlines the requirements for the workflow tool we built. Section 12.3 explains what XML is and give some simple examples of how to use XML. Section 12.4 describes the experiment of creating a workflow tool using XML. Section 12.5 presents the experiences we have achieved from using XML as a basis of a workflow tool. Section 12.6 concludes this paper.

12.2 The Renaissance process model

This section will outline the process model elements found in the Renaissance method description (the Renaissance process model) and how these elements are inter-connected. The description of the Renaissance model gave us the requirements for building the workflow tool and put constraints on what elements we should include. One problem with this description was that the process was described in an in-formal way. This caused that we had to add some elements to the PML to make the process executable.

12.2.1 The basic process model elements

The Renaissance process model focuses on activities and documents needed or produced by the activities. In addition roles are used to assign persons to specific tasks. The following basic constructs were a part of the model:

- **Activity** An activity is decomposable, and consists of sub-activities or sub-tasks. An activity is described by a *name*,*description*, *inputs* and *outputs*. The *pre-* and *post-condition* of an activity is depending on sub-activities/tasks as described in section 12.2.3.

- **Task** A Task is an atomic unit, and cannot be decomposed. A task is also described by a *name*, *description*, *inputs*, *outputs*, *pre-conditions* and *post-conditions*. In addition a task description contains a list of roles *responsible* for the task. A task can be executed in parallel as well as in a sequential manner. Both for activities and tasks, the state of the inputs decides when to execute.

- **Input/Output** Inputs and Outputs refers to documents or collections of document that are involved in the process, and have a *name* as well as a *state*.

- **Roles** Roles define a generalised description of someone responsible for a task (e.g., project leader, secretary etc).

- **User** A user is a named human resource that can play several *roles* in a process.

12.2.2 Relations between process model elements

As indicated in previous section, activities and tasks have relationships to inputs and outputs (documents). In addition activities and tasks can be related in four different ways as illustrated in figure 12.1 and described below:

1. **Consist of relation** describes the relation between an activity and its children (the children can both be activities and tasks).

2. **Sequential activity flow relation** describes that two activities/tasks are executed sequentially.

3. **Concurrent activities relation** describes that two or more activities/tasks are executed in parallel.

4. **Concurrent iterative activities relation** describes that two activities/tasks are executed in a loop until a specified condition is fulfilled.

Since the Renaissance process did not have any conditional process flows (e.g., IF condition1=true THEN do activity1 ELSE do activity2), conditional flow is not a part of the PML.

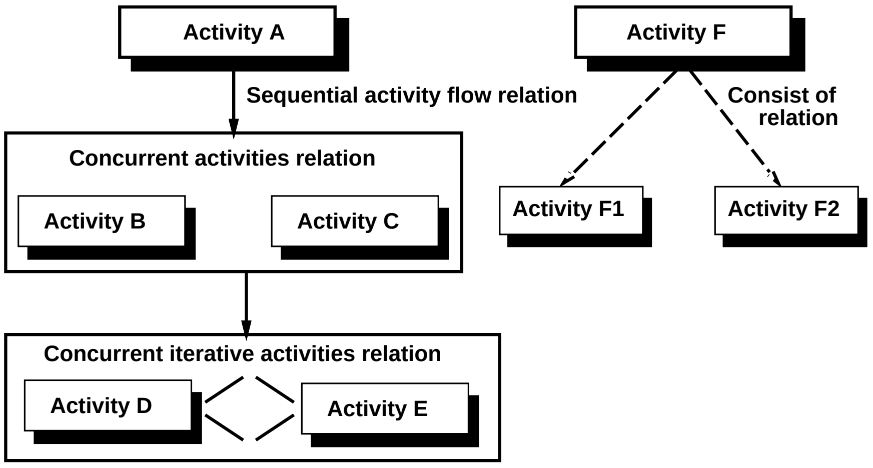

Activity A
Sequential activity flow relation
Concurrent activities relation
Activity B
Activity C
Concurrent iterative activities relation
Activity D
Activity E
Activity F
Consist of relation
Activity F1
Activity F2

Figure 12.1: Relation types between activities/tasks

12.2.3 Process state representation

The description above outlines the PML to be used to implement a workflow tool to support the Renaissance process. More detailed information for how to make this process representation executable was not available from the Renaissance project documentation. We had to decide how to represent process states and find the mechanisms to make the process model executable. We choose to use a state database to cope with the dynamic aspects of the process model. The state database was divided into three parts:

1. **Activity data** Keeps the state information about each activity and task in the process model. Whenever a pre-condition (input) or post-condition (output) is fulfilled, the activity/task state may change. An activity/task can have four states, Not ready (0), Ready (1), Started (2), and Finished (3).

2. **Condition data** Keeps the state information about each condition (pre- or post-conditions). A condition state changes whenever a user has changed the state of a document (read a document, produced a document, coded a document etc). A condition can have three states as Not finished (0), Iterating (1), and Finished (2).

3. **Concurrent data** Keeps track of concurrent activities/tasks (both concurrent and concurrent iterative activities/tasks. Two states are allowed for concurrent state: Concurrent (0) and Not concurrent (1).

A more detailed description of the modelling language and the state database can be found in [Sim98].

12.3 eXtensible Markup Language (XML)

XML is very similar to Hyper Text Markup Language (HTML) in many ways, which is the most popular Web markup language today. HTML has revolutionised the Web, by making it possible for everyone to create hyper-link related document consisting of text, tables, sound and graphics. HTML is very well suited for creating web-pages, but it lacks the capability for specialisation. HTML formats *how* the web-page data will look like, rather than *what* that data represents. HTML is has also only predefined *commands* (tags), and it can therefore not be tailored for specific needs.

XML is more flexible, because you can define your own markup elements. This means that XML makes it possible to tailor the XML documents for different needs, and makes it possible to use XML to represent all kind of data for different purposes. The rest of this section will explain what XML is and how to use it.

12.3.1 What is XML?

Extensible Markup Language (XML) is a specially design subset of *Standard Generalised Markup Language* (SGML), originally simplified and targeted at the WEB. You can use it to format and transfer data in an easy an consistent way. The syntax of XML is similar to HTML, further explained in next sub-section.

12.3.2 Markup Tags

Tags are used as directives to applications reading XML-text and are enclosed text strings in angle brackets for example $< TAG >$. In HTML, these tags are used to tell the web-browser what colours to use, the size of font, to include images etc. In XML, it is up to the application reading the XML-file, what different tags mean. In figure 12.2, a small example shows how XML can be used to store information. The first tag in the figure is a processing instruction that tells the application that this document is an XML document and uses XML version 1.0. The rest of the tags in the example are tags that are defined for this example only. The XML-file above structures the data in a document consisting of one or more customers. To create hierarchical structures, a *start tag*, like $< DOCUMENT >$, and an *end tag*, like $< /DOCUMENT >$ is used. Start and end tags are used to put data in context and to group data. Between a start tag and an end tag you can either put data or you can put more tags to define a multi-level hierarchy.

12.3.3 Document Type Declarations (DTD)

Since there are no restrictions for what tags you can define and how to structure these tags, it is usable to define a Document Type Declaration (DTD). The DTD is not strictly necessary in many XML documents, but to make sure that a document is written correctly

```
<?XML version = "1.0" ?>
<DOCUMENT>
<CUSTOMER>
  <NAME>
    <LASTNAME>Smith</LASTNAME>
    <FIRSTNAME>John</FIRSTNAME>
  </NAME>
  <PROFESSION>Student</PROFESSION>
  <PHONENUMBER>1-800-5412</PHONENUMBER>
</CUSTOMER>
<CUSTOMER>
...
</CUSTOMER>
</DOCUMENT>
```

Figure 12.2: Small XML example

a DTD is used. An XML processor can first read a DTD, then uses this DTD to check if the XML documents follow the structure define in the DTD. To define a DTD you need to define what tags are valid, what order should the tags go in and what tags can contain other tags. You can say that a DTD defines the syntax for XML-files and the XML preprocessor uses this information to find syntax errors in the XML file. To make it easier to under stand what a DTD is, figure 12.3 presents the DTD for the example shown in figure 12.2.

```
<?XML version = "1.0" ?>
<!DOCTYPE document [
<!ELEMENT document (customer)+>
<!ELEMENT customer (name, profession, phonenumber)>
<!ELEMENT name (lastname, firstname)>
<!ELEMENT lastname  (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT profession (#PCDATA)?>
<!ELEMENT phonenumber (#PCDATA)*>
]>
```

Figure 12.3: Example of XML Document Type Declaration

As we can see, the DTD defines what tags are valid, and how the tags are structured. Note that the + symbol means that the item can be repeated one or more times, the * symbol means that the item it refers to can be repeated 0 or more times, and finally the symbol ? means that you can have 0 or 1 element . The example above shows that one *DOCUMENT* can consist of several *CUSTOMER* items, and that a *CUSTOMER* can have several *PHONENUMBER*s.

12.3.4 Tool support

There are several XML tools available on the market today. You can download most of them from a Web-site for free. The functionality these tools provide varies from syntax checkers to full-fledged XML parsers that builds up the document structure for example as Java data-structures. Most XML parsers has support for creating unique identifiers within a XML-file, which makes it easier to refer to elements in the XML document.

Also web-browsers have support for XML. Currently, Internet Explorer 4.0 from Microsoft has support for parsing XML documents and generating a hierarchically structured tree representation. In version 5.0 of Internet Explorer and version 5.0 of Netscape, the support for XML has been expanded.

All the major software companies like IBM, Microsoft, Sun, Adobe, Netscape, AT& T are developing XML tools for creating and parsing XML files. For a list of over fifty XML tool implementation, take a look at this web-page [XML99]. To get an introduction to XML, Steven Holzner's book *XML Complete*[Hol98] is recommended.

12.4 The experiment

Autumn 1998, three 4th grade students at the Department of Computer and Information Science, at the Norwegian University of Science and Technology (NTNU) started to work on a workflow tool that can guide a user through the Renaissance process step-by-step.

12.4.1 Workflow tool implementation

These three students assign the the Renaissance workflow project, worked on the prototype for four months using about 1000 man-hours of work for implementing the system. In this time, they have created:

- A graphical workflow tool, that produces a XML representation of the model [Fug99] as shown in figure 12.4. This tool was created as a Java-applet using standard Java-classes to draw the graphics and generate XML code.
- A workflow engine, that validates the XML-file, parses through the XML-document and read and changes states of the process model. The workflow engine offers a CGI-interface and was implemented in Perl. A C++ XML parser was used to validate and parse through the XML document. The workflow engine supports also cyclic loops in the process [Sim98].
- A web-based graphical workflow client, that guides the users interactively through the process [Bre99]. This tool was implemented as Java applet communicating with the workflow engine through a CGI-interface.

Figure 12.4 shows a screen capture of the graphical workflow tool we made. It is a screen capture of two different windows. The main window named *Applet for renMS project* is the modelling tool consisting of several buttons and a screen-area to draw the model (partly covered by another window). To the right in the screen-area we can see an example of the Renaissance process represented as five activities (the boxes). The other window, named *XML for the Renaissance Method* is the result of pressing on the *Show XML* button and is the process model represented in XML generated by the tool.

Although the prototype is not very stable and advanced yet, we were very pleased that we could do so much in this short period of time and with little resources.

12.4.2 The Renaissance process model represented in XML

We choose to use XML to represent the process model in our workflow prototype. XML was initially chosen, because we wanted to see how well XML was for this purpose and to get an evaluation of practical use of XML.

First we would like to present the Document Type Declaration (DTD) for the description of a task in XML. The DTD was used to define the grammar for our PML as well as making it possible to check syntax and grammar of the process model. Figure 12.5 shows the DTD for the Renaissance process model. Note that the ID listed in the DTD listing generates an unique ID for the whole XML file.

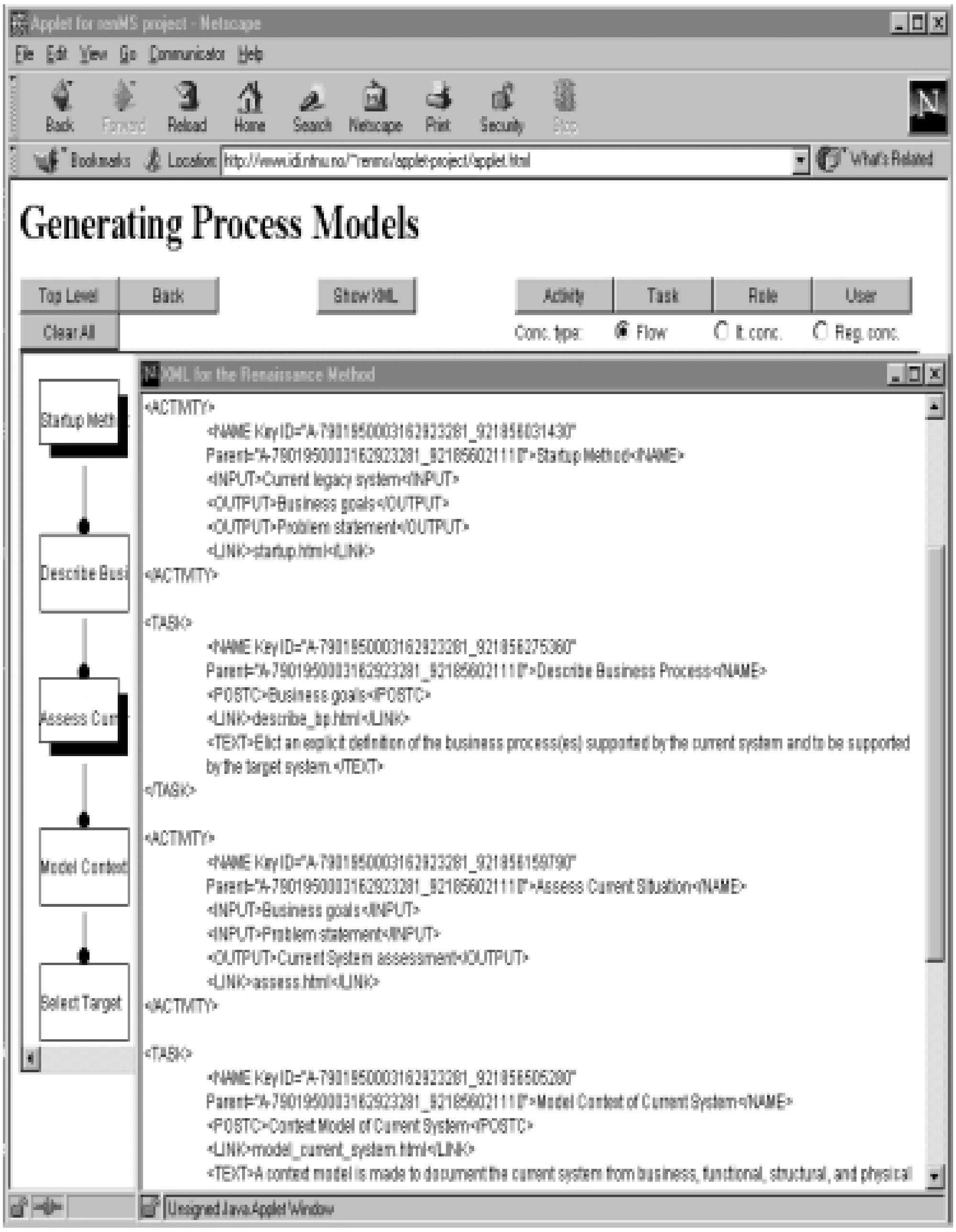
A similar DTD file was also created for *inputs/outputs* as well as for *roles* and *users*. As mentioned above the DTD-files was used to check the grammar and syntax of the XML-files. Another use is to use the DTD file as input for a graphical process modeller tool. What information you can enter the workflow tool is depended on the definition of the PML found in the DTD-files. In this way, it is possible to change the modelling language without changing the tool.

Now it is time to see how the process is represented in an XML-file. In figure 12.6, one activity is described in XML.

12.5 Experiences

In our project of developing a workflow tool using XML technology, we wanted to see how well suited XML was for representing process models. The last PSEE we built in our research group, EPOS [CHLN94, COWL91, NWC97], we used Prolog syntax to represent the process model. Actually, Prolog is not very far from XML when it comes to representation of information, but the syntax of XML is simpler for un-experienced users. We found that the main benefits from using XML were:

1. **XML makes it easier for unexperienced users to model their own process models.** This is mainly because XML syntax is similar to HTML, and that XML is rather


Applet for renMS project - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.idi.ntnu.no/~renmo/applet-project/applet.html
What's Related
Generating Process Models
Top Level
Back
Show XML
Activity
Task
Role
User
Clear All
Conc. type: Flow It. conc. Reg. conc.
Startup Meth
Describe Busi
Assess Curr
Model Context
Select Target
XML for the Renaissance Method
Unsigned Java Applet Window

```
<ACTIVITY>
        <NAME KeyID="A-7901950003162923281_921856031430"
        Parent="A-7901950003162923281_921856021110">Startup Method</NAME>
        <INPUT>Current legacy system</INPUT>
        <OUTPUT>Business goals</OUTPUT>
        <OUTPUT>Problem statement</OUTPUT>
        <LINK>startup.html</LINK>
</ACTIVITY>

<TASK>
        <NAME KeyID="A-7901950003162923281_921856275360"
        Parent="A-7901950003162923281_921856021110">Describe Business Process</NAME>
        <POSTC>Business goals</POSTC>
        <LINK>describe_bp.html</LINK>
        <TEXT>Elict an explicit definition of the business process(es) supported by the current system and to be supported
        by the target system.</TEXT>
</TASK>

<ACTIVITY>
        <NAME KeyID="A-7901950003162923281_921856159790"
        Parent="A-7901950003162923281_921856021110">Assess Current Situation</NAME>
        <INPUT>Business goals</INPUT>
        <INPUT>Problem statement</INPUT>
        <OUTPUT>Current System assessment</OUTPUT>
        <LINK>assess.html</LINK>
</ACTIVITY>

<TASK>
        <NAME KeyID="A-7901950003162923281_921856505280"
        Parent="A-7901950003162923281_921856021110">Model Context of Current System</NAME>
        <POSTC>Context Model of Current System</POSTC>
        <LINK>model_current_system.html</LINK>
        <TEXT>A context model is made to document the current system from business, functional, structural, and physical
```

Figure 12.4: Screen capture from the graphical workflow tool

```
<?XML encoding=’’UTF-8’’>
<!ELEMENT database (activity|task)*>
<!ELEMENT activity (name,
                    (input)*,
                    (output)*,
                    (concurrent)?,
                    (description)?)>
<!ELEMENT task (name,
                (pre-condition)+,
                (post-condition)+,
                (concurrent)?,
                (role)+,
                (user)*,
                (description)?)>
<!ATTLIST activity
          key ID #REQUIRED
          parent IDREF #IMPLIED>
<!ATTLIST task
          key ID #REQUIRED
          parent IDREF #REQUIRED>
...
```

Figure 12.5: The Renaissance process model DTD

readable and easy to understand.

2. **XML makes it easier to create workflow engines, since many XML-parsers are already available.** We found that we could build a workflow engine in a relatively short time, because tool support for parsing the XML and building data-structures from XML-data already were available.

3. **XML makes it easier to make the workflow tool available on the web.** This is mainly because XML-tools are implemented in Java or many XML-tools are easy to integrate with a CGI-server. HTML code is also easy to include into XML documents.

4. **XML makes it easier to create graphical modelling tools.** Since Java is an excellent choice for creating graphical modelling tools (through good graphical support and user-interface support), a Java-based XML processor make the transmission from the Java graphical representation of the model to a XML document easy.

5. **XML makes it easy to change the process model language.** The DTD makes it possible to change the language without changing the whole code for the applications using the XML document. This is possible, since the XML-processor will read the DTD first and then check and build the data-structure for the XML-document.

```
<database>
<activity key=''A0''>
<name>Renaissance method</name>
<input>Current legacy system</input>
<output>Operational target system</output>
<output>New business goals</output>
<output>Revised business process</output>
</activity>
...
</database>
```

Figure 12.6: An activity represented in XML

Although, we were very pleased using XML to represent process models, XML has also one major disadvantages. We found that it was very hard to represent dynamic data using XML. The main reason for this, is that it can be hard and slow to frequently update specific parts of the XML-document (often stored as files in directories). For instance, we did not use XML to represent the states of the process (states of tasks and activities etc.). To do this with XML, we had to read the whole XML-document into a data structure, change some parts of the data structure, and translate the data-structure back to an XML-file. We chose to use small Unix-databases to represent process state, since these databases required small overhead to change its content.

Our experiences with XML indicate that it is suitable for storing and representing the static parts of process models. By static parts, we don't mean parts that will never be changed, but parts that will only be changed once in a while. The DTD will define the syntax of the PML and can be used to check syntax of the XML files. Template process models can also be made by using an almost empty XML-document. It is rather simple to implement tools that create template process models based on process model instances, by removing specific data from a XML document. This can be used later on as a starting point for modelling similar processes.

12.6 Conclusion

In this paper we have looked at how XML can be used to make it simpler to create workflow tools. First we introduced the background for the experiment than founded the requirements for the prototype we built. Although our mission was to create a workflow tool for a specific process using a specific PML, the approaches we used can be used for other similar projects.

XML is really just a way of organising data. What makes XML so useful is that the XML language itself is not fixed and can be tailored to serve different purposes. It is really up to the one building the application to define *what* tags to put in and *how* to organise these tags. In addition you can use Document Type Declarations to define what tags and how tags must be organised if XML files should be used by an application. Most XML parsers has built in syntax and grammar checker, which is a very useful feature when building a program that uses a textual model as input. Another good reason for using XML is that XML documents are relatively easy to understand for software programs as well as for humans. Since many people are familiar with the syntax of HTML, XML requires little extra effort to understand. All lot of tools are available to make the transition between a XML document and a data-structure as easy as possible.

Generally, we think that XML technology is a good help for researchers in general to make it easier to create prototypes need some kind of model representation. At last for our future prototypes, XML will be used.

Acknowledgement

We will give a big thank to Morten Simonsen, Jørgen Andre Brecke and Hans Kristian Fuglenes who have executed experiments using XML to build a workflow tool. In addition, we wish to thank the Renaissance project for giving us an challenging exercise and Mark Greenwood at Information Process Group, University of Manchester, for giving useful feedback on this paper.

Part III

Core Papers

CHAPTER 13

Supporting Distributed Cooperative Work in CAGIS

Heri Ramampiaro[1], **Alf Inge Wang**, and **Terje Brasethvik**

Abstract
This paper describes how the CAGIS environment can be used to manage work-processes, cooperative processes, and how to share and control information in a distributed, heterogeneous environment. We have used a conference organising process as a scenario and applied our CAGIS environment on this process. The CAGIS environment consists of three main parts: a document management system, a process management system, and a transaction management system. The paper describes how these main parts may be configured and used together in order to support cooperative work in distributed environments.

Keywords: Document Modelling, Process Modelling, Transaction Modelling, and Cooperating Agents.

13.1 Introduction

After the introduction of the Internet, more and more projects are taking place in heterogeneous environments where both people, information and working processes are dis-

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway, Phone: +47 73 594485, Fax: +47 73 594466, Email: {heri,alfw,brase}@idi.ntnu.no

tributed. Work is often dynamic and cooperative and involves multiple actors with different kinds of needs. In these settings there is a need to help people coordinate their activities, share documents and information, and to manage access to shared resources. While the web makes it fairly easy to distribute information, the web itself does not contain explicit mechanisms to plan and coordinate activities and tasks, to organise, describe and classify information, or to control access to - and ensure consistency of - project documents.

In the absence of ”web-librarians” that can fulfil such tasks, the users themselves often have to figure out ad hoc solutions for doing this. To help the users, what is needed is a small set of powerful, easy-to-use and flexible tools that may be readily configured to support the task at hand. The CAGIS project - Cooperative Agents in the Global Information Space - aims to support such tasks by using a combination of software agents and small web-accessible tools.

This paper describes how our CAGIS environment, described in section 13.2, addresses the challenges given above. In section 13.3, we present a conference scenario to make the problems and challenges more concrete, and the CAGIS environment is applied to this scenario in section 13.4. Section 13.5 discusses our approach and concludes the paper.

13.2 The CAGIS environment

The CAGIS environment consists of three main components: A system for handling of distributed documents and document understanding, a system for supporting cooperative processes in a distributed environment, and a flexible transaction management system for shared, distributed resources. The following three sub-sections will describe our effort in these research areas in more details.

13.2.1 Document models and tools

Documents published on the web have to be organised, classified and described to facilitate later retrieval and use. One of the most challenging tasks is the semantic classification - the representation of document contents. This is usually done using a mixture of text-analysis methods, a carefully defined (or controlled) vocabulary or ontology, as well as a scheme for applying this vocabulary when describing a document. The CAGIS document model toolset helps the users of a project group to do this semi-automatically, by way of a domain model expressed in a conceptual modelling language and by using text analysis tools as an interface to perform the actual classification and search. In other words,we use a conceptual model as a basis for creating meta-data descriptions (figure 13.1). These meta-data descriptions may then be accessed through our java-model viewer that enables search and browsing of documents through a standard web browser environment.

Fundamental to our approach is the use of a conceptual modelling language to define and

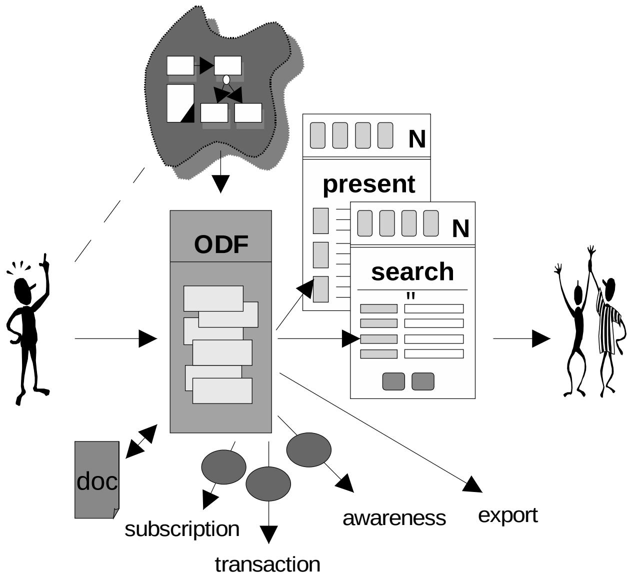


Figure 13.1: Conceptual modelling for meta-data descriptions

visualise the domain specific vocabulary to be used in the classification and retrieval process. Conceptual modelling languages contain the formal basis that is necessary to define a proper ontology, yet at the same time they offer a visual representation that allows users to take part in the modelling, and to read and explore documents by interacting directly with the models. The conceptual modelling language may thus be used throughout the entire process of classifying and retrieving documents on the web. In our approach, we use the Referent model language [Arn98] an ER-like language with strong abstraction mechanisms and sound formal basis.

Our approach may be described as a three-step process: ***Domain Model Construction, Document Classification*** and ***Browsing & Retrieval*** - outlined below.

Domain Model Construction:

Conceptual modelling is mainly a manual process. However, our domain models must be related to the text of the documents to be classified, hence we use a textual analysis tool as input for the modeling. A reference set of documents from the domain is run through a word frequency analysis tool, which produces a list of high frequency terms as input candidates for the actual conceptual modelling task. This is a manual and cooperative task performed by a selected set of users. Concepts are carefully selected, related to each other and given a textual definition. In order to prepare the finished domain model for later document classification, we then add lexical linguistic information to the model , i.e. the model is enhanced by adding a term-list for each of the concepts in the model. The term-list is a list of synonyms, instances and and conjugations for each concept that will be used later in the classification of a particular document.

Document Classification:

Documents are classified by selecting domain model fragments that reflect the document content. This is performed semi-automatically by matching the document text against the term-lists for each of the concepts in the model. Concepts found in the document are then shown to the user as a selection in a graphical model viewer and the user may manually refine the classification by selecting and deselecting concepts and relations. When the user is satisfied, the selected model fragment is translated into XML and is stored as an ”Object Descriptor File”. The user also has to provide a selected set of properties for the document, such as its author, title etc. These attributes are also stored within the ODF.

Browsing and Retrieval:

In order to retrieve documents, the users enter a natural language query phrase which is matched against the conceptual model in a similar way as in the classification process. The domain model concepts found in this search phrase (if any) are extracted and used to search the stored document descriptions. Users may then refine their search by interacting with the model. Found documents are presented as list in a Web-browser interface. We also have an enhanced ”document reader”, that is, when reading a document, all the terms in the document that matched a model concept is marked as a hyper-link pointing to the definition the model concept.

The layered architecture of our document tool is shown in figure 13.2. As mentioned, the main parts of the system are the web-enabled user interface and a set of servlets running on a standard web-server.

- The user interface is centred around a Java-based Referent-model viewer. As mentioned, users may interact with the model, explore concept definitions and relations, and then use the viewer directly in order to perform both classification and retrieval.
- The Java servlets define the overall functionality of the system. They are invoked from the model viewer and coordinate the linguistic tools incorporated into the system.
- At an ”intermediary” layer, between the servlets and the web-server, we use a number of linguistic tools analyse natural language phrases and give the necessary input to construct domain vocabularies and classify and retrieve documents. The Word frequency analyser from WordSmith is a commercially available application for counting word frequencies in documents and producing various statistical analyses. A Finnish company, Lingsoft, has two tools for analysing nominal phrases and tagging sentences needed for the classification and retrieval of documents. A smaller Prolog application for analysing relations between concepts in a sentence is being developed internally at the university.

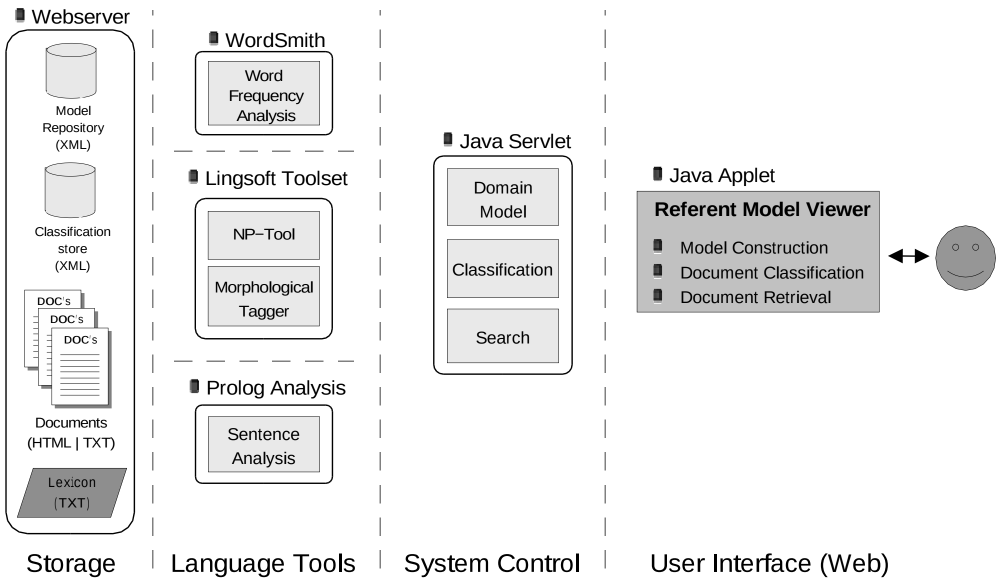


Figure 13.2: Overview of system architecture

- Finally, the documents and their classifications are stored as files at the web server in HTML/TXT and XML format respectively. The domain model is also stored in XML and must be maintained separately. The linguistic tools rest on lexical information that is partly stored within the model XML file and partly integrated with the tools themselves.

A more detailed presentation of our approach and the system is given in [Ter99, Ter00].

13.2.2 Process models and tools

A prototype of a process centred environment (PCE) has been developed to give process support to distributed, cooperative processes in CAGIS. The CAGIS PCE consists of three main components:

Workflow System supporting Distributed Mobile Processes

This workflow system is used to model simple, repeatable workflow processes, and the system offers an agenda-browser for the end users. The workflow system allows an instantiated workflow model to be distributed as several process fragments on different workspaces. One benefit of this is the possibility to adapt the workflow to local environmental conditions. The workflow model instances are defined as XML-files distributed over several workspaces, and can be modified by the owner of the workspace. The ability to move and change workflow instances during enactment, can be used for reallocation of activities, dealing with exceptions (someone responsible for a particular activity acts sick), and delegation of work. The Workflow system is implemented in Perl, providing a CGI-interface through a web-server. For a more detailed description, see [Wan99, Wan00b].

The Process Modelling Language (PML) for the workflow system defines a process as set of activities that can have mutual pre-order relationships specified in XML syntax. That is, an *activity* can specify a set of *pre-links* identifying what activities to be executed before, and *post-links* identifying activities to be executed after the activity current activity. The pre- and post-links can be written as URLs, and therefore allow the process to be distributed over several workspaces. Every activity definition specifies a code part (a script). This code part is expressed in HTML, and can be used to simply present information, to specify a user input through a form, or to start a Java-applet.The term *process fragment* is used to name a group of activities in a workspace as part of the whole process. A process fragment is specified by a name, a workspace (location), and a list of references to activities.

Software Agents to support Dynamic, Cooperative Processes

While the workflow system described above takes care of simple, repeatable process, we use software agents to support more cooperative and dynamic processes. Software agents typically takes care of inter-workspace (inter-group) activities such as negotiation activities (e.g., about of resource allocation), coordination of artifacts and workflow elements between workspaces, brain-storming, voting, marked support (e.g., agents as buyer and sellers of services), etc. Our multi-agent architecture consists of four main elements:

- **Agents** An agent is set up to achieve a modest goal, characterised by autonomy, interaction, reactivity to environment, as well as pro-activeness. We have identified three main types of agents: (1) *Work agents* to assist in local production activities, (2) *Interaction agents* to assist with cooperative work between workspaces, and (3) *System agents* to give system support to other agents. Interaction agents are mobile, while system and work agents are stationary.

- **Agent Meeting Place (AMP)** AMPs are where agents meet and interact. AMPs support agents in doing efficient inter-agent communication. There can be different types of AMPs for different purposes. Each AMP will have a defined ontology (the framework described in section 13.2.1 can be used here), which the agents have to follow. We can perceive special AMPs for negotiation, coordination, information exchange, selling and buying services etc.

- **Workspaces** A workspace is a temporary container for relevant data (artifacts, models etc) in a suitable format to be accessed by tools, together with the processing (work) tools. It can be private, as well as shared. Files stored in a repository can be checked in and out to a workspace.

- **Repositories** Repositories can be global, local, or distributed, and are persistent storage of data. Experience Bases are one specific type of repository, that we can use in our multi-agent architecture to support community memory.

The multi-agent architecture is implemented in Java, using IBM Aglets framework to provide mobile agents, KQML is used for inter-agent communication, and ORBIX CORBA is used to offer communication to other applications and other agent systems. More detailed description of the multi-agent architecture can be found in [WLC99, PHBN99, HN00].

Agent-Workflow GlueServer

The Agent-Workflow GlueServer provides interaction between the workflow system and the multi-agent system. A **glue model** in XML defines the relationship between workflow elements and software agents. The **GlueServer** will offer services for a workflow activity to trigger an agent and vice versa. The GlueServer is implemented in Java, and ORBIX CORBA is used to facilitate communication with the agent system and workflow systems. More information about the GlueServer can be found in [WCL00, Bjø00].

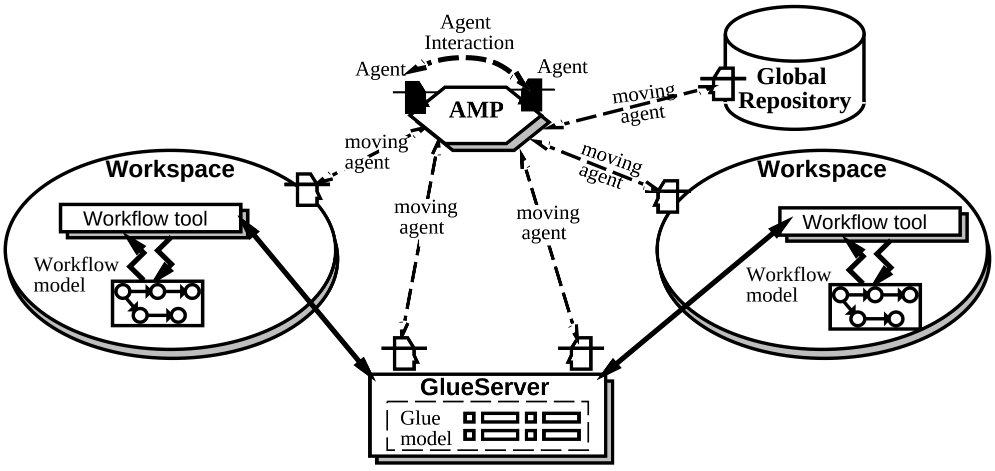


Figure 13.3: The CAGIS Process Centred Environment

Figure 13.3 shows a simplified illustration of how the different components in the CAGIS PCE interact. In figure 13.3, there are two workspaces, each running a workflow tool (engine) with a local workflow model. In reality, this workflow tool can be shared, and the local workflow models in the two different workspaces can have relationships between them. The figure illustrates two different ways that software agents can interact with workspaces. In the first way, the agents can interact directly with the user in the workspaces, using a graphical user interface to configure and interact with the agents.

In the second way, all interaction with software agents goes through the GlueServer and the workflow tool. The workflow tool can activate an agent, or an agent can activate the workflow tool. The figure also shows that agents can be used to access repositories, but workspaces can also access files in the repository directly (not shown explicitly in the figure).

13.2.3 Transaction models and tools

A transaction is a basic work unit (or possibly a program segment) executed to perform some function or task by accessing and manipulating a shared database. Transaction modelling aims to capture the essential characteristics of transactions. In general, these characteristics include transaction behaviour and applied constraints.

Transaction modelling in CAGIS was motivated by the assumption that database management systems (DBMSes) will be used to manage resources. The main purpose of a transaction management system is then to control and manage access to shared resources, and to make sure that this access is done in accordance with prespecified consistency or correctness preservation constraints. This section briefly describes our effort in developing a transaction framework and a transaction management system for cooperating agents.

The transaction specification framework

We have proposed a transaction framework to specify and execute customisable transaction models [RN00].

The main purpose of this framework is to provide configurable, application-specific transaction models. The ability to adjust the required degree of control to be provided through the transaction models is crucial in order to cover different situations. Some situations may, for instance, require strict control for data correctness, others may see control as just a burden, and so on.

The framework support has two parts: *Transaction characteristics specification* and *transaction execution specification*.

The former characteristics specification defines the main properties of the transactions to be executed: *ACID properties*, *relationship among the involved transactions* (e.i., transaction structures and transaction dependencies), *adopted correctness criteria* (i.e., user/application dependent criteria) and *applied policy* (i.e., rules for what mechanisms are to be used and how they are used). These characteristics are statically defined and must be done before the designated transactions are executed. The latter execution specification defines how the transaction execution is to be performed at run-time, in terms of composition of *management operations* (e.g., delegate, abort, commit etc.) and *regular access operations* (e.g., read, write etc.). The execution specification must conform the former, and has some fixed (though tailorable) initial operations, while the remaining operations can be adjusted at run-time.

Thus, CAGIS transaction framework distinguishes between static and runtime dynamic specifications. Compared with related frameworks, such as ACTA [CR94], ASSET [BDG$^+$94], and TSME [GHM96], the main difference is on the dynamic property. Such a property is important since it is not always possible to predict all aspects of application in advance. This becomes particularly relevant when taking software agents as well as cooperative work into consideration.

A detailed presentation of this transaction framework is given in [RN00].

The transaction management system

A system for the transaction specification described above is depicted in figure 13.4. This system is divided into two components; a specification environment and a runtime management system.

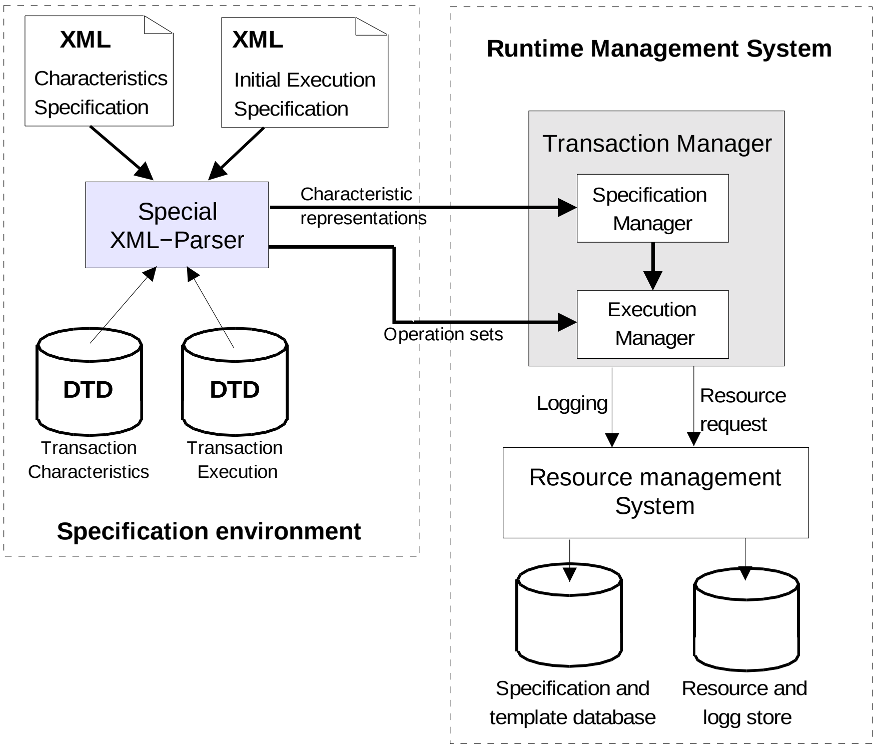

XML
Characteristics Specification
XML
Initial Execution Specification
Special XML−Parser
Characteristic representations
Operation sets
DTD
Transaction Characteristics
DTD
Transaction Execution
Specification environment
Runtime Management System
Transaction Manager
Specification Manager
Execution Manager
Logging
Resource request
Resource management System
Specification and template database
Resource and logg store

Figure 13.4: Transaction management architecture

- **The specification environment** allows a transaction model designer to specify the characteristics of a transaction, and a set of operations to be run by a transaction.

Both specifications (characteristics and execution) are done in XML. A special XML parser is used to verify specifications, against a corresponding prespecified Document Type Definition (DTD). This parser transforms (1) the specification of characteristics into an internal representation, and (2) the execution specification into a set of internal operations. They are both used by the transaction manager (see below) to control and monitor transaction executions. Further, both specifications are kept in an internal specification database. To avoid redundant specifications, the specification environment allows the designer to browse and check whether the given transaction characteristics are already in the specification database. This implies that future adjustments can be performed without needing to do the specification from scratch. Finally, if changes are made, the designer is asked whether the current specification shall be saved as a new version or to replace the old one.

- **The runtime management system** consists of a transaction manager and a resource management system. *The transaction manager* is responsible for managing the specification and execution of transactions, and to ensure that any transaction execution is done according to the specified characteristics. For example, if an ACID model is chosen, the transaction manager will enforce atomic and isolated execution. Correspondingly, if the atomicity property is relaxed, then the same manager will ensure that any failure would not necessarily cause global rollback. Instead, partial abort can be issued. As shown in figure 13.4, the transaction manager again consists of a specification manager and an execution manager. *The specification manager* is responsible for controlling that all necessary representations from the specification environment are complete. In other words, it has to make sure that the specification of transaction characteristics can indeed be supported and that all necessary semantics are represented. If such a specification is not fully satisfiable, it will either notify the designer and ask him/her to adjust the specification, or it will choose a closest supported specification that can be found in the specification database. Otherwise, it makes the characteristics information available to the *execution manager*. The latter manager ensures that a transaction is executed consistently with respect to its stated transaction characteristics and execution specifications. Based on these specifications, the execution manager issues the necessary and suitable transaction management operations. This means, that it issues `begin`, `abort` or `commit` operations, and other management operations that the user has specified.

  The resource management system is responsible for managing and providing system resources for those actually executed transactions. It also maintains execution information of running transactions, and uses this to handle transaction aborts and system recovery. Finally, the resource management system is responsible of making sure that committed results are kept in a persistent store.

  The transaction management system described above was implemented in a working prototype [Sel00, KK00] based on Java and the IBM Aglet-workbench. It has served as a test-bed for the transaction specification framework.

13.3 Conference Scenario

This section describes briefly a conference organising process which we shall use as a scenario. This based on the scenario presented in [OSVS82]. The seven main activities of the conference organising process are shown in bold-face in next paragraph.

First the Program Chair will initialise the conference management process by **Planning and announcing the conference**. People wanting to contribute to the conference will submit their papers. PC members will later **Record submitted papers** as well as information about the authors. Then, **Reviewers will be chosen** based on their expertise, and the **Paper reviewing** starts. The PC members will then **Collect reviewing results**, and a (electronic) review meeting will be held to **Determine acceptance of papers**. Accepted papers will then be **Grouped into sessions** and a final program, including a time-table for conference sessions, will be produced.

In this paper, we will focus on the last main activity, *Group Accepted Papers into Sessions*. Figure 13.5 shows the two main sub-activities of this activity.

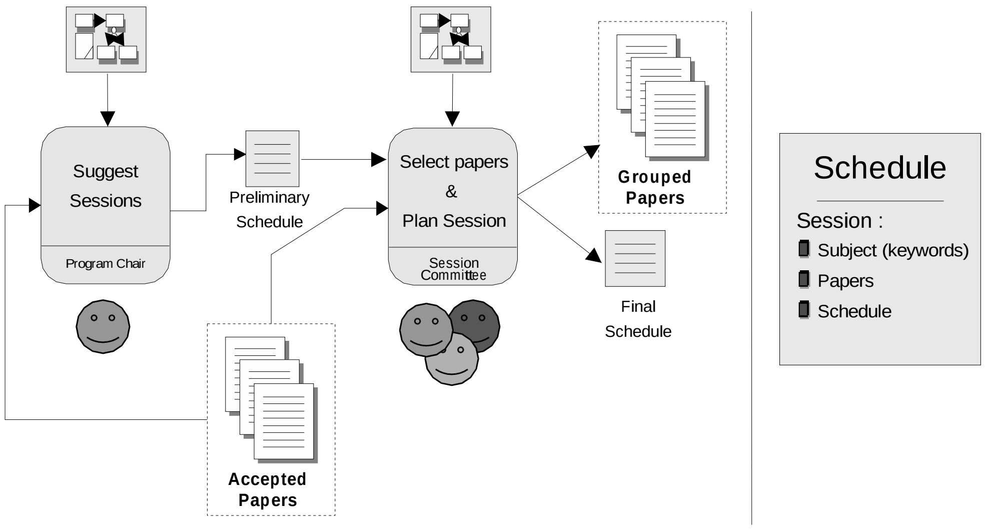


Figure 13.5: Grouping of Accepted Papers into Sessions

Suggest Sessions

The Program Chair is responsible for this activity, which can be decomposed into the following process steps:

1. Match all papers against a document model defining terms and expressions, and the relationships between them for the research domain.

2. Suggest a session division according to subjects.
3. Create a preliminary session schedule.
4. Set Up Session Committees (from Program committee members), one for each committee.

Select papers & Plan Sessions

All members of Sessions Committees are responsible for this activity, and it can be decomposed into the following process steps:

1. Determine session subject & goals: An *initial* session description will contain session subject and goals.
2. Check papers for session: Sessions Committee members should mark papers that are relevant for a session to notify their interest. Papers will be marked *"possible"*.
3. Paper allocation: If papers are marked by more than one Session Committees, these committees must negotiate about which session is going to get the paper. Papers finally allocated to a session will be marked *"taken"*.
4. Check timeslot for session: Each session committee will mark the timeslot for the session.
5. Session allocation: Sessions that have the same timeslot will have to negotiate. When all sessions are allocated, the result will be added to the session description. The session description will now have the state *"final"*.
6. Publish session description: Each Session Committee will publish their session description to the other Session Committees and Program Chair.

In this paper we assume that the Program committee members will be distributed on different locations and the work with organising the conference will be done through computer interaction (without any physical meetings).

13.4 The CAGIS environment applied on the scenario

This section outlines how the CAGIS environment, consisting of tools and models to support documents, processes and transactions, can be applied on the scenario described in section 13.3. Our suggested architecture to support this scenario is shown in figure 13.6.

The two main activities we are focusing on, **Suggest Session** and **Select papers & Plan Session**, are executed by the Program Chair and the Sessions Committees respectively. In our solution we have therefor chosen to model the scenario using one workspace for Program Chair and one workspace for each Session Committees. Each workspace has a local process defined in a process model, and a workflow tool that enacts this process

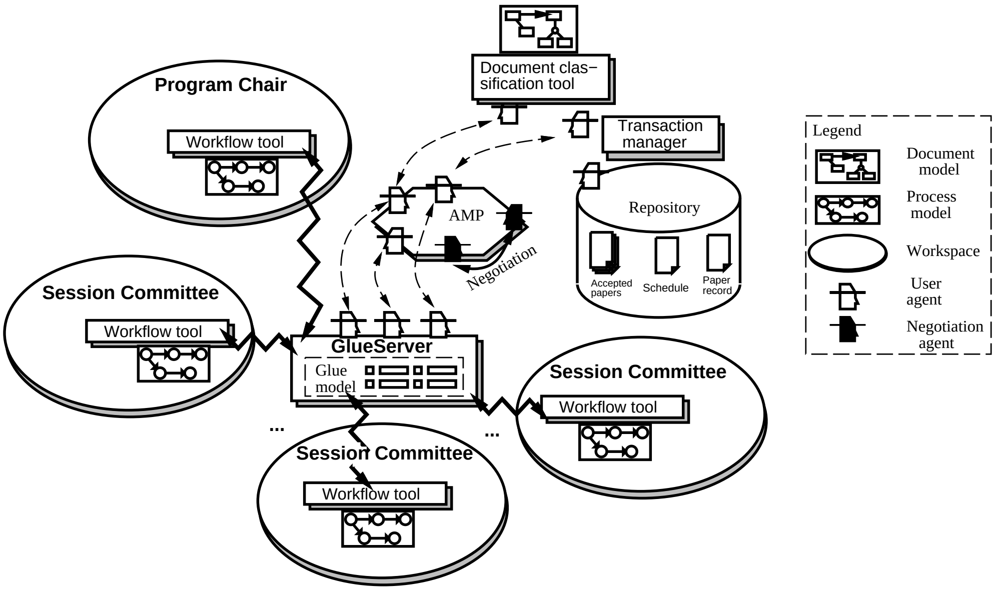


Figure 13.6: The CAGIS framework applied on the scenario

model. The process models are defined according to the process steps defined for **Suggest Sessions** and **Select papers & Plan Session** as given in section 13.3.

The Program Chair's first process step is to classify all papers according to their themes. This is done by using the **Document classification tool** to match all papers against the domain model. The domain model defines the vocabulary of keywords, extracted from the preliminary conference topics and from the submitted papers. The matching of papers against the domain model is visualised in the document model viewer, thus illustrating how the papers are thematically related to the concepts in the domain model. The Program Chair may interact with the model viewer to achieve the proper subject division of papers.

The ***Workflow tool*** will here notify the ***GlueServer***, that will initialise a *document agent* that may be used to access the documents through the document servlet. In the two next process steps, the workflow tool will present the Program Chair with necessary documents and tools for creating a preliminary session schedule and setting up session committees. When session committees are selected among PC members, the session committee members will be notified through email describing what session committee to attend and what workspace to access.

The Session Committees will then start to work in their workspaces according to the process model enacted by the workflow tool (remember that all conference organising work will be done distributed on computers). First they have to determine session subject & goals, the workflow tool will notify the GlueServer that will initialise *brainstorming agents* for each session committee member. The result of this brainstorming process will end as an initial session description written by session chair. The next step of the process will be for the session committees to choose what papers to be in the session. Here the workflow tool notifies the GlueServer to initialise *paper select agents*. The paper select agents will retrieve information about available papers, and let the session committees mark interest of papers. The paper select agents will then mark papers in the *Paper record* in the repository (see figure 13.5). The result from marking papers will be returned to the GlueServer. If papers have been marked by several session committees, negotiation agents will be initiated to negotiate about which session is going to get the paper. If the negotiation process goes into a deadlock, the Program Chair will be notified, and he/she will make a final decision. The next step, is for the session committee to mark a timeslot for the session. This process works exactly like paper selection, but *session selection agents* will be used instead. When all session committees have selected their timeslots, the final session description is published to all participants, and the final conference program can be produced.

The **transaction manager** is responsible for managing the integrity of the document in the repository, and to ensure that agents always leave the system in a consistent state. Based on the description above, the Session Committees share both the schedule document and the paper record. Therefore, conflicts are likely to occur. There are several possibilities to resolve these conflicts. For example, one may provide an exclusive lock for each access, thus prohibiting other to see any changes until the related process is finished. This is usually unacceptable, since it might delay the session arrangement process. Another possibility is to permit reading access. This allows other committees to see the intermediate changes, and therefore eases their decision process. A third solution is to permit simultaneous updates (i.e., write/write conflict). However, achieving consistency is possible only if the system supports multiple-version handling, with a sophisticated merging mechanism to capture all possible changes.

Next, tasks to support the session selection process are assigned to agents. As mentioned, this may involve document access too. To allow the transaction management system to ensure consistency, all agent operations involving repository access are managed as part of transaction execution. This also ensures that conflicting document access is managed properly. Moreover, suppose that a transaction consisting of several agent operations is initiated by the GlueServer. Then, assume that one of the involved agents fails, for example, while selecting papers from the paper record. Using traditional ACID transactions, this would cause a global rollback, that would discard all changes made so far, and kill all associated agents. However, if a lot of effort has already been invested, restarting all tasks from scratch may be expensive. To cope with this, we model each agent operation as a subtransaction of that executed by the GlueServer. Therefore, instead of aborting the transaction and killing all involved agents, the transaction manager allows the failing agent to just undo some of its changes. Other agents may proceed as normal.

13.5 Conclusion

This paper has presented the CAGIS toolset and its applications to a conference organisation scenario. The CAGIS toolset consists of a set of separate tools that may be used together to provide support for cooperative work across the web. The three major components of CAGIS are the Workflow tool, the Document classification tool and the Transaction manager. Each of these tools is implemented in true Web style, i.e. they are built around a standard Web server and use XML as a data storage and interchange format. These tools may all be configured according to the actual situation and use. The *Workflow tool* allows for the creation of individual workspaces to support the activities of the workflow, in addition, the workflow tool offers the ability to enact the part of the process model that the workspace supports. The *document classification tool* uses a domain specific vocabulary, the domain model, to help users classify and search for documents. The *transaction tool* offers support for the specification and execution of customised and application-specific transaction models. The transaction manager thus offers the ability to design the required correctness criteria and to define and execute transactions that enforce these criterias on shared resources. Central to our system, and binding the individual tools together, is a GlueServer. The *GlueServer*, configures a set of software agents that can activate the different CAGIS tools. The *glue model* defines the relations between the individual workflow elements that may reside in different workspaces and the software agents that may be used to access the individual tools. This way, the various components of the CAGIS toolset may be used together in order to provide situation specific cooperative support.

Our CAGIS environment is not only applicable to conference organisation processes. The CAGIS environment can be used to support any process where people are working together, and where people and information are distributed. Examples of such processes can be cooperative software engineering processes, distributed educational processes, distributed organising processes, processes of selling and buying merchandises on the web etc. All these processes are characterised by distribution of people and information, and require people to interact and cooperate to reach the goal of the process.

Furthermore, when combining the tools in the CAGIS environment, we enhance the functionality of one specific CAGIS tool. In [Alf00], an example of how the document models and tools can enhance the CAGIS multi-agent architecture is given. The document models and tools are here used to model the agent ontology, which define the language software agents can speak. In [Wan00c], the transaction models and tools (in this paper called workspace manager) offer a way managing consistency of changing workflow models. This means that the CAGIS environment offers a selection of tools, which can be used in different combinations to give specific support. Future work will investigate more thoroughly what tools to pick for different scenarios.

Acknowledgements

We would like to thank Reidar Conradi for his constructive comments. The CAGIS project is sponsored by the Norwegian Research Council's Distributed Information Systems (DITS) Programme.

CHAPTER 14

A Multi-Agent Architecture for Cooperative Software Engineering

Alf Inge Wang, Reidar Conradi[1], and **Chunnian Liu**[2]

Abstract
This paper looks at how Cooperative Software Engineering (CSE) can be supported. We first investigate the process aspects by presenting a traditional process architecture supporting CSE. Then we propose a multi-agent architecture for CSE, which is better in terms of simplicity and flexibility, and particularly useful in modelling and providing support to cooperative activities. We describe an industrial scenario of CSE, and show how to apply the proposed architecture to this scenario. The scenario is based on a software development and maintenance process for a Norwegian software company.

Keywords: *Computer-Supported Cooperative Work, Cooperative Software Engineering, Software Process Technology, Multi-Agent Systems*

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7035 Trondheim, Norway. Phone: +47 73593444, Fax: + 47 73594466, Email alfw/conradi@idi.ntnu.no

[2]Beijing Polytechnic University (BPU), Beijing, P.R. China. Chunnian Liu's work is partly supported by the Natural Science Foundation of China (NSFC).

14.1 Introduction

Most of the work in the software process community has been focusing on how to make people work together in an organised and planned way (partly pre-planned). For high-level processes with little details, it is likely that it is possible to make people work in this manner. However, the development of software involves people that cooperate to solve problems and to do actual work. These kind of processes are very hard to support by traditional software process support tools, because the focus will be more at cooperative aspects than pure coordination of work [WLCM98b]. In this paper we introduce an architecture to provide support for cooperative software engineering.

Computer-Supported Cooperative Work (CSCW) is a multidisciplinary research area focusing on effective methods of sharing information and coordinating activities. CSCW systems are often categorised according to the time/location matrix [Gru94] (synchronous / asynchronous and non-distributed / distributed). We may add an extra dimension to the CSCW typologies, considering different kinds of cooperative work in the order of increasing complexity of the process support they need [LC98]:

- **Ad-hoc cooperative work** such as brainstorming, cooperative learning, informal meetings, design work, *etc.*. Process modelling support here is implemented through awareness triggers.
- **Predefined/strict workflow**, in traditional Office Automation style represented by simple document/process flow. Examples of such systems can be Lotus Notes [Orl92], Active Mail [GSSS92] and MAFIA [LvRH90].
- **Coordinated workflow**, such as traditional centralised software maintenance work consisting of check-out, data-processing, check-in, and merge steps. There exist several systems supporting coordinated workflow (mostly prototypes), e.g. EPOS [CJM92], MARVEL [BSK95] and APEL [DEA98].
- **Cooperative workflow**, such as decentralised software development and maintenance work conducted in distributed organisation or across organisations. Here the shared workspace and the cooperation planning are the main extra factors from the process point of view. Example of a system supporting distributed organisations and processes is Oz [BSK95].

By Cooperative Software Engineering (CSE) we mean large-scale software development and maintenance work which falls into the last two categories in the above list. Because of the rapid spread of World Wide Web as the standard underlying platform for CSCW systems, more software companies are moving from the traditional centralised working style to the decentralised one. In the decentralised CSE, communication, negotiation, coordination and collaboration among the various participants are more complicated, because people are not only geographically distributed, but may also work on different platforms, at different times, with different process models. A better understanding about CSE processes is needed as well as a full range tool support of such processes. The research in this area will help the software industry to change the work style to take full advantage of WWW, and will enrich the research area of Software Process Technology (SPT) in which so far a centralised work style has been often assumed implicitly.

Compared with the traditional architecture (as found in systems similar to EPOS [WLCM98b]), an agent-based architecture is advantageous in terms of simplicity and flexibility, and particularly useful in modelling and providing support to cooperative activities [CMM97, CM96]. In this paper, we try to integrate the areas of CSCW and SPT in a multi-agent architecture. First we investigate the process aspects of CSE by presenting a traditional process architecture supporting CSE. Then we propose a multi-agent architecture for CSE, which is an extension and specialisation of the more general architecture [MDP98] for all CSCW. This architecture will then be applied to an industrial CSE scenario.

14.2 A Traditional Process Architecture supporting CSE

The key issues of CSE are group awareness, concurrency control, communication and coordination within the group, shared information space and the support of a heterogeneous, open environment which integrates existing, single-user applications. All these are related to the software process. Within the SPT community there have been research on each of the issues, but from a slightly different point of view (see, for example [COWL91, NWC97, CHL95, Jac96]). To see how CSE is supported by a traditional architecture, we present the process support in such a architecture. This architecture is usually realized by a **Process-sensitive Software Engineering Environment, a PSEE**, with a spectrum of functionalities and associated **process tools**. The support is needed in three main areas; at the Process Modelling template level (defining process in PML), at the Instance level (adding detail to process model), and for Enactment and monitoring.

To full fill the goal as a process support architecture for CSE, some underlying components are needed. These components makes it possible to apply the architecture in a distributed, heterogeneous environment. *First*, a portable platform infrastructure for PSEE-based client tools are needed (candidates are HTML/CGI and Java). *Second*, the architecture must offer an integrated environment for tool operation and communication (candidates are CORBA [OMG97] or DCOM [CRW96]). *Third*, we need facilities for distribution of tools and workspaces (candidates are CORBA or DCOM). *Fourth*, we need a community memory or experience base to store template process models (a candidate is Experience Factory[BCM$^+$92]).

Figure 14.1 presents a general PSEE architecture for CSE. Its cooperative support is provided through a shared workspace where files and parts of the process model are stored and shared. The private workspaces are provided with tools for planning, scheduling and enaction of the process model. The shared workspace provides support for cooperative planning and negotiation, and for coordination through cooperative protocols. The shared workspace is managed by a project manager. The architecture is web-based, and repository and experience base support is provided through a web-server and a CGI-interface to

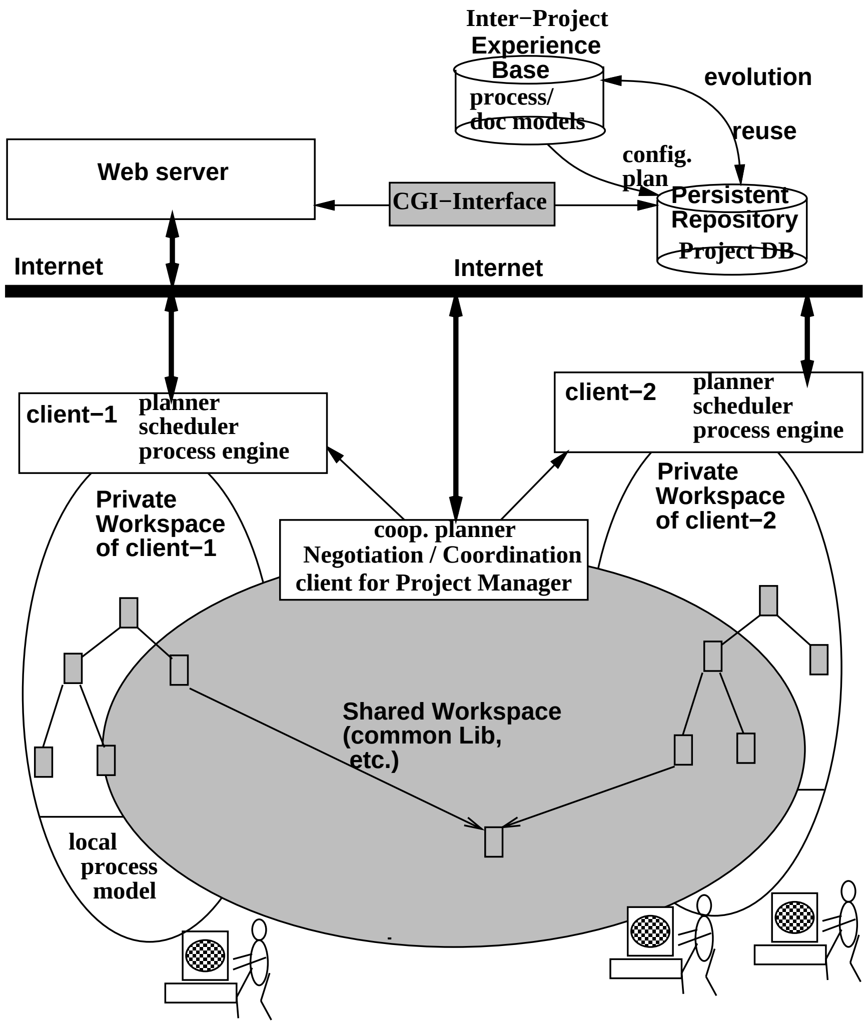


Figure 14.1: A General Process Architecture Supporting CSE.

the repositories.

There are, however, several problems with such a PSEE/CSE architecture. *First*, it is too centralised and has too much flavour of a centralised database surrounded by a fixed number of applications. *Second*, too homogeneous models are used assuming one common PML. This means that one PML must be used by all involved partners, although this is no necessarily the best solution. *Third*, it is hard to change process tools and models. Due to the distributed and open setting, we should allow dynamic reconfiguration of process models, as well as for process tools. *Forth*, a open-ended spectrum of process tools may be needed to offer better support for cooperative work, than what classical PSEE architecture can offer.

As will be seen in the next section, a multi-agent architecture seems more appropriate for a general CSE.

14.3 Multi-Agent Architecture for CSE

The previous section shows how complex a CSE environment could be. Similar situations exist in other areas such as Distributed Artificial Intelligence, Business Process Management, and Electronic Commerce. Nowadays, it is believed that the Multi-Agent Systems (MAS) are a better way to model and support these distributed, open-ended systems and environments. A MAS is a loosely-coupled network of problem solvers (agents) that work together to solve a given problem. The main advantages of a MAS are:

- *Decentralisation*: being able to break down a complex system into a set of decentralised, cooperative subsystems. In addition, many groups of organisations are inherently distributed.
- *Reuse of previous components/subsystems*: That is, building a new and possibly larger system by interconnection and interoperation of existing (sub)systems, even though they are highly heterogeneous. Thus, we do not request a common PML, so different PMLs can be used in different subsystems.
- *Cooperative Work Support*: being able to better model and support the spectrum of interactions in cooperative work, since software agents can act as interactive, autonomous representatives of humans.
- *Flexibility*: being able to cope with the characteristic features of a distributed environment such as CSE, namely incomplete specification, constant evolution, and open-endness.

In the remainder of the paper we try to model the problem area CSE as a MAS consisting of four components **Agents**, **Workspaces**, **Agoras** and **Repositories**.

14.3.1 Agents

In this paper, an **agent** is a piece of autonomous software created by and acting on behalf of a user (or some other agent). It is set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to environment, as well as pro-activeness. The whole process (and meta-processes) of CSE is carried out by groups of people, using tools, such as production tools, process tools, and communication tools. Each participant can create a set of software agents to assist him/her in some particular aspects. There are also some *system agents* created *by default* for the administrative purpose in the architecture. We can perceive the following types of agents:

- *System agents*: These cover default agents for the administration of the multi-agent architecture, such as creation and deletion of agoras. In some cases more specific system agents are needed. *Monitor agents* track events in workspaces and agoras in order to collect relevant measurements according to predefined metrics. *Repository agents* can provide intelligent help for searching for information.

- *Local agents*: To assist in work within local workspaces. These agents act as personal secretaries dealing with local process matters such as production activities as well as to define, plan, and enact process models.

- *Interaction agents*: To help participants in their cooperative work. Such agents can be viewed as shared process agents, and they include four subclasses. *Communication agents* are used to support a spectrum of more high-level communication facilities. All information flow, also simple communication, uses agents as the underlying communication mechanism to make the architecture clean and simple. *Negotiation agents* verbalise their demands (possibly contradictory) to move towards an agreement through the process of joint decision making [Mul96]. *Coordination agents* support, e.g., a project manager issuing a work-order that involves a group of developers; or a higher-level manager being called in to mediate between negotiating agents to reach an agreement. *Mediation agents* are used to help negotiating agents reach an agreement. In doing so, mediation agents may consult the Experience Base (EB, cf. Section 14.3.4), act according to company policies, or ask a project manager (human) for help to make decisions.

14.3.2 Workspaces (WS)

A **workspace** is a place where human and software agents access shared data (usually files) and tools which can be private or shared by a group of people. In addition, interaction between users and software agents takes place in workspaces. The simplest form of a workspace can be a file-system provided with services to read and write files. A more advanced workspace can provide file versioning, access to of some repository, awareness services, web support etc. BSCW [BHT97] is one example of an advanced web based workspace implementation. Agents can access data in the workspace either directly or indirectly through tools.

14.3.3 Agoras

An **agora** [MDP98] is a place where software agents meet and interact , but can also be a market place where agents “trade” information and services. Agoras should provide agents with more intelligent means to facilitate their interaction. The main purpose of the agora is to facilitate cooperative support for applications and agents. Below we propose the following preliminary functionalities that any agora should support.

1. *Inter-Agent Communication*:
   This is not simple information-passing, it rather conveys intentions, goals, beliefs, and other mental states to form the foundation of negotiation and other complex interactions. An agora should facilitate agents to announce their capabilities and to get in touch with agents capable of doing specific tasks. The following services are needed to facilitate inter-agent communication:

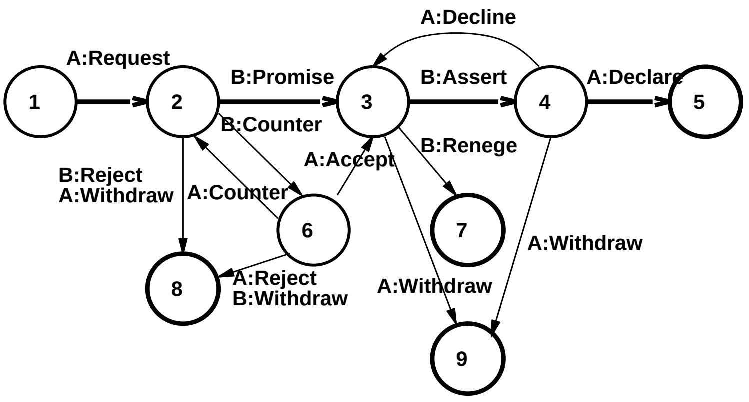


Figure 14.2: An example of speech-act

- An agora should provide a predefined set of speech-acts [WF86] (conversation types), such as *proposal, counter-proposal, acceptance, rejection, confirm, deny, inform etc*. The various speech-acts types will define how agents can interact with each other. In many cases a speech-act is represented as a state transition diagram as the one shown in figure 14.2. The speech-acts act as shared process model for how agents should interact. Figure 14.2 shows states of a conversation between two agents A and B. States are shown as circles while transition between one state to another is shown as arcs. The terminal state 5 indicate a successful conversation and the bold lines shows the path of a successful conversation.

   - An agora should specify a common *syntax* for messages transmitted through the agora, so that the recipient can analyse the contents of a message.
   - An agora should specify a common *semantic* of an agent language. One part of this semantic is defined through speech-acts, i.e. what state transitions of a conversation that a message will cause. The semantic also ensures that software agents interpret the same words similarly.
   - An agora should specify pragmatics for agents. This means that agents shall not lie to other agents and the agents intentions should be honest.

2. *Inter-Agent Negotiation*:
   The progress of a negotiation depends mainly on the negotiation *strategies* employed by the agents involved, but agoras should provide mechanisms to minimise communication overheads.

14.3.4 Repositories

In our architecture, a **repository** represents an information server that in the simplest form only provide services to store and retrieve persistent data. A more advanced repository will provide services for data modelling, searching through data, comparing data, computing data etc. Repositories can be accessed either by tools or by agents.

The most fundamental repository is the production repository storing versioned products. Other repositories may include process models, experiences, user-error-reports etc. The more advanced repository is the **community memory** across projects which can be realized by an Experience Base. Stored information from previous projects can then be used to create more accurate estimates, foresee problems, and improve processes for new projects [NWC97].

14.3.5 The CSE Multi-Agent Architecture

Within our architecture, the four CSE components are interconnected and interoperate as follows:

1. **Agents are created by people** to help them work; by other agents to perform delegated work; or by default to manage workspaces or agoras.

   Note that agent creation is a process of instantiation of the corresponding agent classes based on templates.

2. **Agents are grouped mainly according to people grouping**. In CSE, we can usually perceive various groups of people working as a team. The mechanism we have used to group people and agents is by workspaces. Shared workspaces are used to group teams of human and software agents working together, while private workspaces provide support for one human and possibly several software agents.

3. **Interaction between agents is via agoras**. Agoras can be to provide agent interaction between group workspaces as well as interaction between private workspaces. Some system agents are created by default to manage the agora (creation, deletion, and bookkeeping).

4. **Agents uses repositories**. There are monitoring agents, to perceive events in workspaces and agoras, to collect relevant data and to store the data into repositories. In this way, the community memory is built. And in decision making, or when some negotiation runs into difficulties, *mediation agents* can help by utilising previous experience from some repository.

5. Within a group of agents and their shared workspace, any existing process models are allowed, and the traditional process architecture described in Section 14.2 can be applied. On the other hand, we can also apply this agent-based architecture recursively to a group of agents.

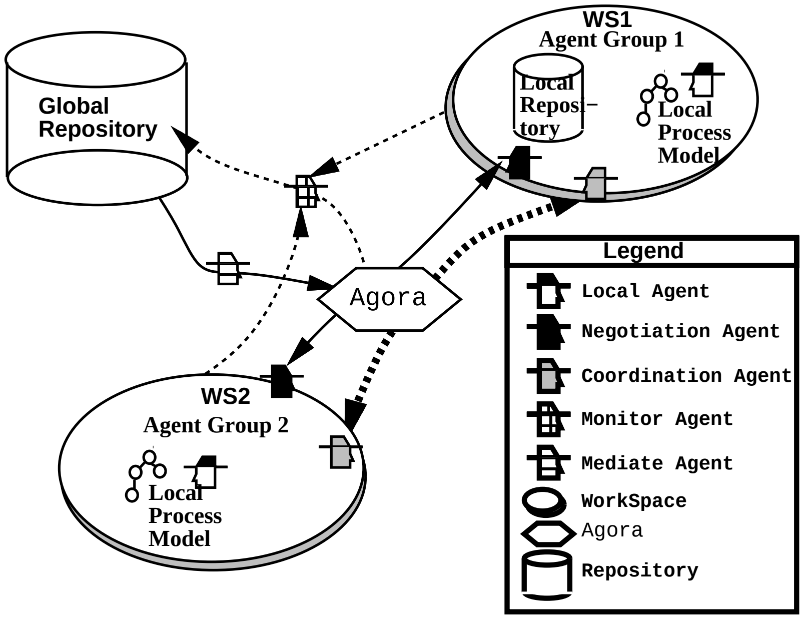


Figure 14.3: Multi-Agent Architecture for Cooperative SE

Figure 14.3 shows the four components of the CSE architecture and their interconnection and interoperation. Note that the figure shows different types of agents and repositories. In section 14.5 we will see a concrete architecture when our approach is applied to a CSE scenario.

14.4 An industrial scenario

This scenario is based on the software development and software maintenance process from a real Norwegian software company, in this paper called AcmeSoft. The company's products exist on various operating system platforms, including Microsoft Windows NT and various UNIX platforms. In this scenario we by *software development* mean the development of future releases and updates of products, whereas by *software maintenance* we refer to the correction of defects in released software. Common to these processes are a *production and testing process* which builds the products for requested platforms and the *delivery process* which creates the distribution media and ships products. An overview of the main activities in the scenario process is shown is figure 14.4. Corresponding responsible groups are listed below the activity name.

The **Development process** focuses on work that is directly related to changes of software products and the planning and scheduling of these changes. The three main process steps are: 1) *Release and update planning*, 2) *Scheduling*, and 3) *Implementation*.

The **Maintenance process** is triggered by a one of the following maintenance reports; *Software Query Reports* (SQRs): Error report or desired, *Release Problem Reports* (RPRs): Internal problem reports, or *Production orders*: Requests from customers for a given product or product revision.

The maintenance agreements define priorities system for SQRs and RPRs, with five levels from Critical down to May_not_be_implemented named P0 to P5. Based on this classification, the correction phase of SQRs and RPRs is divided into the five following process steps: *1) Registration* (by the development department), *2) Estimation of resources* (which developer, effort, *3) Sendout* (send SQR/RPR to developer), *4) Correction* (actual problem fix done by developer), and *5) Module testing* (by developer).

The **Production and testing process** starts after a freeze in development code or after defect corrections, or when customers request a delivery revision built for a specific platform. The process consists of the three following steps: *1) Production, 2) Testing*, and *3) Verification*

The **Delivery process** consists of activities to store products on distribution media and to ship products. This process is initiated when a product release or update is made available, and on customer demand. The delivery process can be divided into two main activities: *1) Delivery* and *2) Shipping*.

14.5 Application of the Architecture to the Scenario

Figure 14.5 shows our multi-agent architecture for the scenario described in the previous section. In this architecture there are six agent groups (workspaces) corresponding to the six groups First Line Support, Maintenance Process group (MPG), Update/Release Planning group (URPG), Development group, Production and QA group, and Delivery

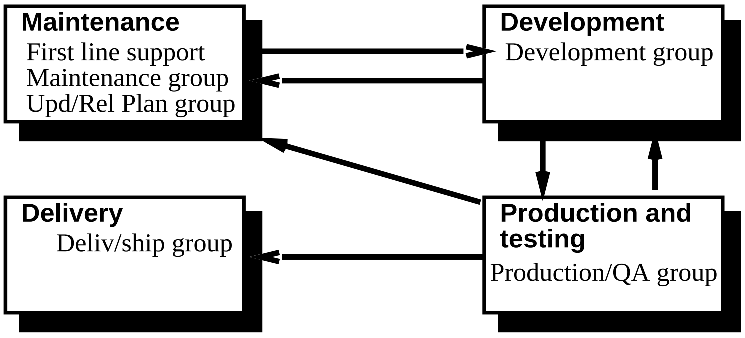


Figure 14.4: Scenario process

and Shipping group.

Each group has their shared workspace. Each group has also its own process model, which may be an existing legacy one. The process models of different groups can be heterogeneous. Furthermore, some groups may have this new agent-based architecture recursively. That is, an agent group may be spilt into several (sub)groups, and the shared workspaces into several (sub)workspaces. In all cases, the inter-(sub)group communication is modelled explicitly via agoras. In the architecture, we can observe different kinds of interaction agents belonging to various groups. Examples are:

- Two **negotiation agents**, one belonging to the First-Support group, the other to the MPG, communicate via the agora *Ag1*. Because when the First-Support Office conveys a user request for a change and the desired deadline for the new revision, the MPG may or may not authorise the changes (according to their configuration control policy). Even if the planning office agrees to authorise the changes, a deadline need to be negotiated. In other words, the Defect Priorities (P0–P4) shown in figure 14.5 are the result of negotiation, rather than a simple information passing.

- Two **negotiation agents** (one belonging to the URPG, the other to the MPG) communicate via the agora *Ag2*. See below for detailed discussion.

- Some **coordination agents** are observed in between the MPG (or the URPG) and the Development Team. This means that the change-order are given to a group of developers. So the MPG needs to coordinate the development work . The change-order will actual cause changes to the local process models. In this perspective we can see the coordination of a change order as a process model change.

- **Other** negotiation and/or coordination agents could be observed in the figure, but for simplicity we just show them as simple communication agents.

AcmeSoft has a distributed repository used as an EB of the company. The EB holds information about previously completed projects and products and about previous up-

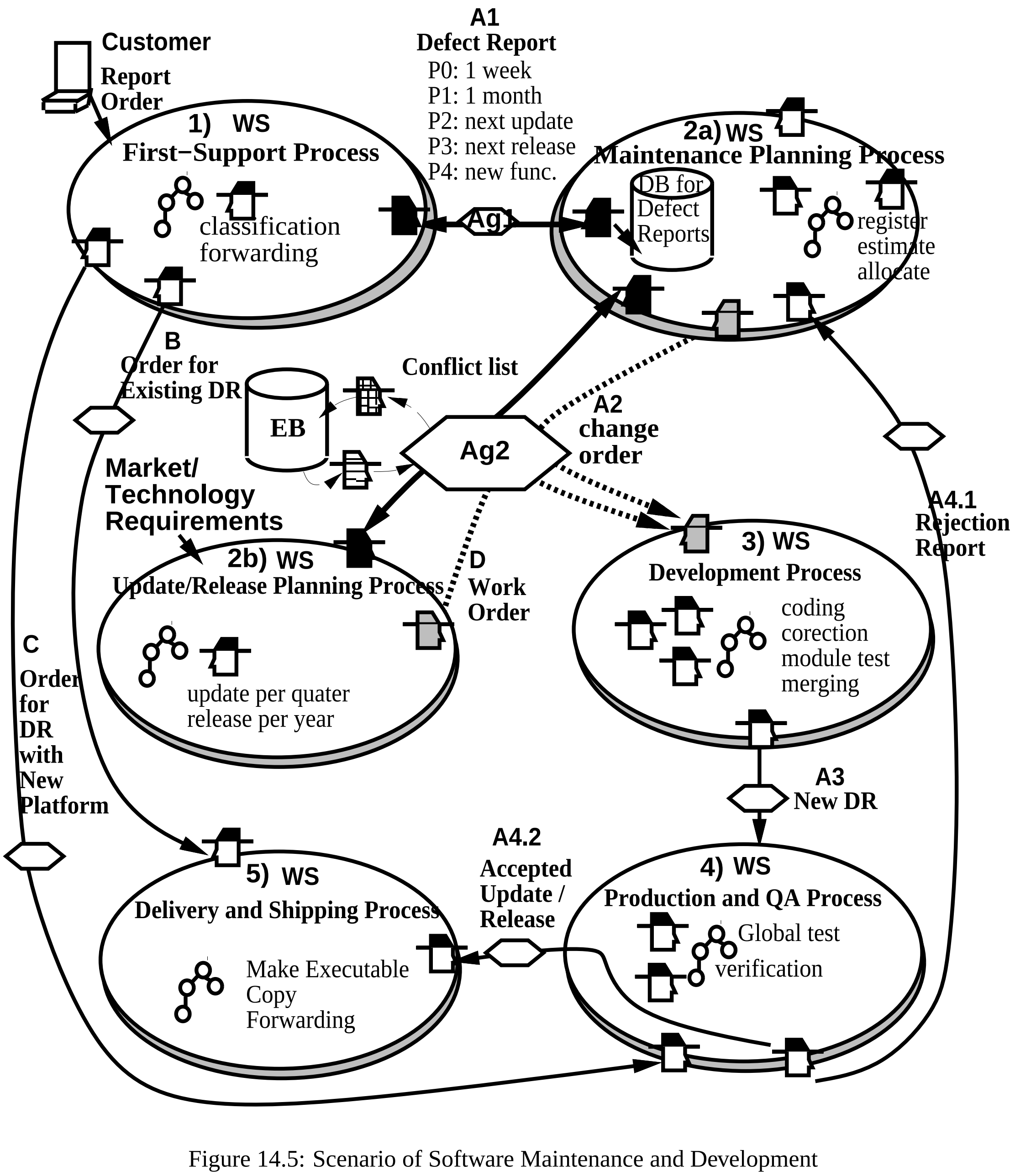


Figure 14.5: Scenario of Software Maintenance and Development

dates/releases of the current products. Typical data are: the project profiles, evolution patterns, performance metrics, and process models.

Let's have a closer look at the agora Ag2. There are various inter-agent activities occurring in or transmitted through it. In the following, we explain some of these activities:

1. **Negotiation and coordination** between the URPG and the MPG.

   First, remember that the main task of the URPG is to plan the next update and the next release of a company's products. In doing so, the URPG should make decisions on issues such as what defects should be fixed and what new functionalities should be included in the next update/release. What to include is based on market analysis and feedback from users (prioritised defect reports). Naturally, market and technology analysis contributes to this decision-making. The relevant information is presented in users' defect reports with the priority P2–P4, which is received, analysed, and stored by the MPG. Based on this information, the MPG would give requests, suggestions or advice to the URPG about the contents of the next update/release. On the other hand, the URPG may accept, reject, or negotiate these proposals. All these inter-agent activities are carried out through the agora M2.

   Secondly, remember that the same development team is responsible both for maintaining existing products and for developing new updates/releases. Here we have a conflict in resource allocation, and negotiation is necessary.

2. **Mediation** in the negotiation between the URPG and the MPG.

   As indicated, in solving a resource allocation conflict, the MPG and the URPG may not by them selves be able to reach an agreement. E.g., the MPG would like to "lend" programmer **A** to fix an error in a product for user **B** that demands an immediate reaction. On the other hand, the URPG would like to have **A** as the chief programmer the full next year for a planned update/release. The problem is that each negotiating agent views the issue only from its own angle, based on local experience. Furthermore, in such a real-life domain, it is hard to evidence that algorithm-based negotiation strategies alone can solve such problems reasonably. Human intervention by a manager of the company may be necessary. How much human intervention the agent will need, depends of the definition of the agent. In this way it is possible to tailor the agent to the needs of the company. It should be possible to state, e.g., that resource negotiation for more than a certain amount of money must be done through human interventions. The mediation agent works on behalf of the higher-level manager, in order to propose an overall beneficial solution. In doing so, the mediation agent may utilise previous experiences by searching the EB. For example, if the EB shows that user **B** has been an "important" customer in the past, the mediation agent may stand by the MPG and persuade the URPG to consider another choice as chief programmer.

14.6 Conclusions and Future Work

In this paper we have introduced an agent-based architecture to solve CSE problems. This architecture consists of four main components: Agents, Workspaces, Agoras and Repositories *Agents* provide flexible and dynamic support to cooperating users, as well as help for doing every-day work. Agents can easily respond to a changing environment (learn, adopt based on experiences in the ExperienceBase etc.). It is widely accepted that real software processes evolve over time, so our process support must adopt and cope with such changes. To enable agents to cope with process changes, they will need to learn from prior experiences. In our architecture this is introduced through *repositories* (ExperienceBases) as well as agents can learn on their own. *Agoras* and *workspaces* are introduced to support agent interaction and grouping of agents, respectively.

Our architecture has been applied on one specific scenario. We believe, however, that our multi-agent CSE architecture is applicable on various situations, processes and organisations. One concrete example is to support meta-process activities, such as discovering/planning process models, negotiation about the process model and the real-world model and assignment of resources to a instantiated process model. Our architecture's main contribution is to give process support where traditional SPTs often fail in respect changing environment and unexpected events. Further work with describe formalities, implementation of prototypes, and experiment with more industrial scenarios will show if this is the case. Another outcome of our research will be to identify disadvantages of MAS.

Acknowledgement

The authors of this paper want to first of all to thank Mihhail Matskin, Sobah Abbas Petersen, Monica Divitini, Heri Ramampiaro for suggestions and discussions regarding the paper and a big thank to Torgeir Dingsøyr and Joar Øyen for reading through and giving useful comments on this paper.

CHAPTER 15

Design Principles for a Mobile, Multi-Agent Architecture for Cooperative Software Engineering

Alf Inge Wang[1], **Anders Aas Hanssen**[2], and **Bård Smidsrød Nymoen**[3]

Abstract
The paper describes experiences we have achieved from implementing a mobile multi-agent system for cooperative software engineering, based on the Aglets technology from IBM. When implementing the mobile multi-agent system, we faced problems dealing with locating agents, inter-agent communication, registration of agents etc. Based on our experiences, we present some design principles for how to locate agents, how agents should communicate, how to manage connection to the agent system, how to register agents and agent places, how to move agents, how to remove agents, and how to give CORBA-agent interaction support. These design principles should be applicable for others wanting to design mobile multi-agent systems using the Aglets technology.

Keywords: Mobile agents, Multi-agent architecture, Software design.

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway. Phone: +47 73 594485, Fax: +47 73 594466, Email: al@finge.com
[2]BEKK Consulting AS, Palekaia 1, 0150 Oslo, Norway, Email: anders.aas.hanssen@bekk.no
[3]Mogul.com, Kongensgt. 51E, N-7012 Trondheim, Norway, Email: BardN@numerica-taskon.no

15.1 Introduction

There are a number of mobile agent technologies available today, both commercial and as research prototypes. Many of these agent technologies only provide a framework for building a mobile agent system, and do not offer extensive solutions for e.g., inter-agent communication, keeping track of agents, locating agents, etc. In this paper we describe some design principles we have extracted from experiences building the mobile multi-agent system called DIAS, based on the mobile agent technology Aglets from IBM. DIAS is one important component for an architecture that supports Cooperative Software Engineering. Agents are used to represent actors in a cooperative effort, and give the users of the agent system support for doing efficient negotiation, cooperation and exchange of data.

When we started building our mobile multi-agent system, we had only a high-level architecture consisting of agents and agent places (more on this in section 15.2). We wanted to allow both stationary and mobile agents, where the mobile agents could move between agent places. User clients to the system can connect and disconnect to the agent system dynamically, while the rest of the agent system can run continuously. As we started to do low-level design, we discovered that mechanisms for dealing with mobility of agents, and dynamic user client connections were not directly supported by the Aglets framework. These mechanisms have been carefully designed to cope with scalability, security, and stability. The rest of the paper describes the results we gained from our low-level design of the DIAS system.

In [Yar98], Aridor and Lange report several design patterns they have found when creating mobile agent applications. In general, agent design patterns are used to capture good solutions to recurrent problems to make agent applications more flexible, understandable, and reusable. The patterns Aridor and Lange present, focus on agent travelling, agent tasks, and agent interaction. *Travelling patterns* give solutions for routing agents among destinations, forwarding newly arrived agents automatically to another host, and using tickets to encapsulate quality of service and permissions needed to move an agent. *Task patterns* are concerned with the breakdown of tasks and how these tasks are delegated to one or more agents. *Interaction patterns* are concerned with locating agents and facilitating their interactions.

Mobile agents have become a very popular research topic lately. Many technologies facilitate moving objects between hosts. Voyager [Gla99] developed by ObjectSpace is a product family consisting of an ORB and an application server supporting mobile agents. Grasshopper [IKV00] is an agent development platform launched by IKV in 1998. It enables the user to create agent applications enhancing electronic commerce application, dynamic information retrieval, telecommunication services and mobile computing. Grasshopper is completely implemented in Java giving the benefit of high distributed integration. Jumping Beans [Ad 99] builds on the Java platform and provides a framework for Java programs to “jump” from computer to computer. The Jumping Beans architecture is based on client-server architecture, where programs moving from one host to another must do this through a central management server. The central management server has a very strict security system. By using a central management server, the Jumping Bean framework is best fitted for small distributed networks rather than WANs (bottleneck).

15.2 The Distributed Intelligent Agent System (DIAS)

This section is a short introduction to the Distributed Intelligent Agent System (DIAS) which is a part of CAGIS Multi-Agent Architecture for Cooperative Software Engineering. [WLC99] The CAGIS architecture uses agents to represent co-operative participants in a cooperative effort and supports coordination, negotiation and communication through agents. DIAS provides a foundation for creating a mobile multi-agent system through high-level agent API and multi-agent services. The DIAS architecture is based on a more general multi-agent architecture for supporting a distributed information technology application [MDP98].

15.2.1 DIAS components

We have tried to keep the DIAS architecture simple and it consists of only two main components. By combining these main components, we get a fully functional mobile multi-agent system. The main components are:

- **Agent** A piece of software acting on behalf of a user. The agent is set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to the environment, as well as pro-activeness. There are three main groups of agents in DIAS: *System agents*, *Participation agents*, and *User agents*. System agents administrate the DIAS architecture. We have identified the following types of system agents:
  Manager agents are responsible for managing Agent Meeting Places (AMPs), *Facilitator agents* facilitate communication between agents, *Monitoring agents* log events and manage security in AMPs, *Repository agents* retrieve information from, add information to, and query repositories, and *Interface agents* provide and interface between the agent system and other applications.

  Participation agents can either be stationary or mobile, and they provide system support for cooperative processes in agent places. Typical participant agents are:
  Communication agents that bring messages or data from one agent to another agent situated on a different agent place, *Negotiation agents* that help other agents to reach an agreement, *Mediation agents* that will act on behalf on higher management and use prior experience to solve locked negotiation processes, and *KQML agents* that make it possible to directly send a KQML message from a user to agent. One advantage with the latter agent, is the ability to communicate directly with other agents without having to implement a specific agent for this purpose.

  User agents are created by a user or by a vendor of agent applications, and can be either mobile or stationary. The DIAS agent API is used by the developer to create user agents.

- **Agent Place** An agent place is where software agents meet and interact in the DIAS architecture. The agent places can be distributed on different hosts, and facilitate means for efficient inter-agent communication based on KQML [FLM97]. Agent places can also have CORBA-support, facilitating external applications to communicate with the agent system through interface agents. The OMGs standard, Mobile Agent System Interoperability Facilities (MASIF), is used to provide a bridge between CORBA applications and DIAS. There are two main types of agent places:
  - *Agent Meeting Place (AMP)* This is the place where the agents advertise their capabilities, communicate with other agents. AMPs are where agents that represent different users can come to and interact. An AMP can be addressed as a DIAS Service Provider (DSP) if an Agent Docking Place (ADP) is connected to it. The AMP will then provide services to the ADP.
  - *Agent Docking Place (ADP)* This is the user-client where the user interacts with his/her agents, and where agents can be created and killed. Agents can also communicate locally in ADPs.

The DIAS architecture was implemented in Java using Aglets Software Development Kit from IBM [LO98] to provide agent mobility. *Only KQML-layer* in JATLite [JPC00] was used to support the KQML communication language in our architecture. Since the contents of a KQML message can be anything, we have chosen to use XML for this purpose. XML gives us an easy way of representing and wrapping data, and there are several XML tools available for Java as well.

15.3 Design Principles for Mobile Agent Systems

This section describes the adopted principles in designing a mobile multi-agent system. A more detailed description of our design principles and how these principles are implemented can be found in [HN00], while an overview of the technology and high-level design issues in DIAS is discussed in [Wan00a].

15.3.1 Agent Location

In a mobile agent system, there most be a mechanism to locate mobile agents. A user agent will start at the users ADP, and can visit to different agent places on different hosts. Participant agents are also mobile and can move between agent places on behalf of a user request or another agent. In DIAS, the ability to communicate with these mobile agents is essential. Thus, a mechanism to locate mobile agents is needed. We have considered three alternatives for locating agents in DIAS:

1. Each agent knows where other agents are situated.
2. Agents make footprints where they go.
3. Agents report to an AMP where they go.

The *first alternative* will require agents to hold much updated information. This will cause problems with performance and scalability, since agents can be mobile. The *second alternative* implies that agents leave footprints to agent places when they are moving to another agent place. When an agent wants to locate a certain agent, it must therefore follow the footprints of this agent. This option can be quite complicated to implement and is not especially efficient. Especially, when agents are killed, all the footprints must be followed and deleted. If an agent has made a far journey, this removal of footprints can be a time and resource consuming process. In DIAS, we have used the *third alternative* to solve the agent location problem. In this approach, an agent must inform his new location to an AMP before moving. We use the term *DIAS Service Provider* (DSP) to name where an agent should report its localisation. By using this alternative, the process of updating agent's locations is distributed to several AMPs. The agent will only inform its new location to the DSP when before moving to another agent place. Replication of DSPs can also be used to achieve better availability of the agent system. Two rules are used to decide the DSP of an agent:

1. Agents created in an AMP will get the current AMP as its DSP.
2. Agents created in an ADP will use the DSP of this ADP. When an ADP is created, an AMP must be specified as DSP. AMPs are used as DSPs because they are persistent while ADPs can be temporary.

All agents have their DSP as a part of their agent ID, making it possible to locate agents' DSPs from an agent ID

15.3.2 Agent Communication

Because it was very hard to combine JATELite's message router with Aglets, we had to create our own agent communication mechanism. When agents communicate, the DIAS system has to locate where the receiver of the message is situated. If communicating agents are at the same agent place, they can exchange messages locally. If communicating agents are on different agent places, the message will be carried by a communicating agent. Here is a simple example showing Agent A at AMP1 sending a message to Agent B located at ADP1. Note that system agents (not shown in the figure) are used to bring the message from Agent A to Agent B.

These steps describe figure 15.1:

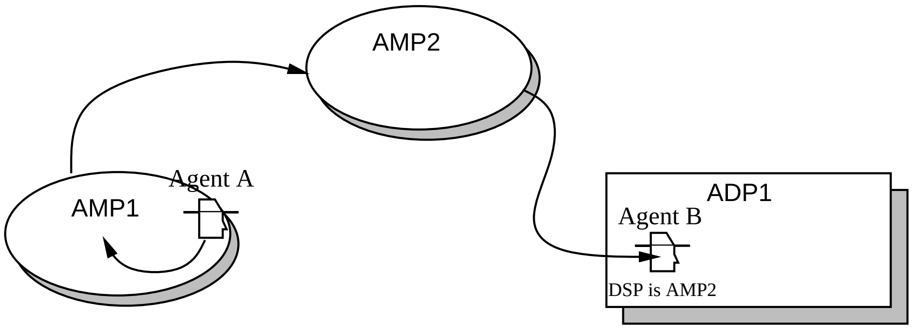


Figure 15.1: Communication example

1. The message from Agent A to Agent B is first sent to a system agent locally at AMP1.
2. A system agent at AMP1 finds the DPS address of agent B, and sends the message to this DSP (here AMP2).
3. A system agent at AMP2 looks up the location of Agent B, and sends the message directly to Agent B on ADP1.
4. Agent B replies the message with an acknowledgement (sent directly back to Agent A)

As the example above illustrates, the DSP of the receiving agent is used to find this agent. AMPs are dynamically exchanging ID information about agents in the system. AMPs that also are DSPs, hold information about agents where-about. There are two types of agents that are important when communicating in DIAS:

- **Facilitator Agent** Every AMP and ADP has a facilitator agent. All messages in the DIAS system have to be sent through the facilitator agent, deciding if a message should go directly to the receiver agent (locally) or via communication agents to find the correct remote receiver.

  Figure 15.2 illustrates two examples for how facilitator agents are used in DIAS. First in figure 15.2, Agent A want to send a message to agent D. The facilitator agent in AMP1, forwards the message using a communication agent to AMP2. The facilitator agent in AMP2 then forwards the message to AMP3, where AMP3's facilitator agent gives the message to the requested agent D. The second example in the same figure is when agent B wants to send a message to agent C. The facilitator agent can then just give the message to agent C, since both are located in AMP2. The facilitator agent will act according to available receiver agent information, which is the agent identifier, agent's DSP identifier and ontology. If the agent and DSP identifiers are missing, the facilitator agent tries to find an agent with matching ontology.

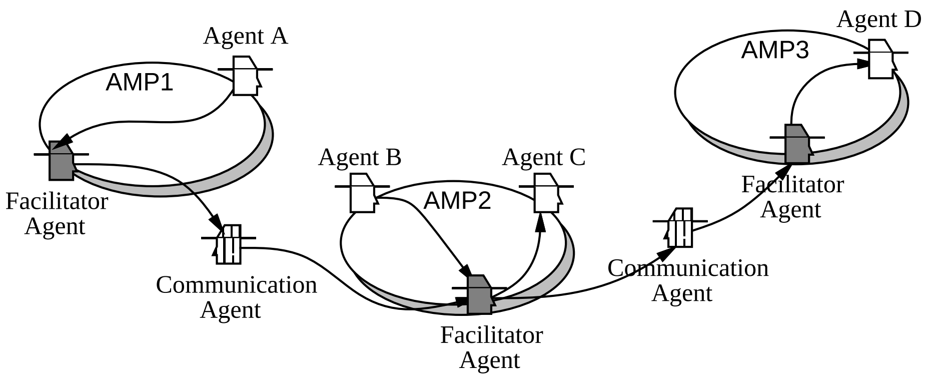


Figure 15.2: An example of how the facilitator agent works

- **The Communication Agent** The communication agent is responsible for bringing a message from one agent place to another. A communication agent is triggered by a facilitator agent, carries the message to the target destination, and delivers the message to facilitator at the target agent place.

15.3.3 Connection of agents and agent places

Users of the DIAS are not connected to the agent system all the time, therefor mechanisms for handling user connection and disconnection are needed. When a user wants to connect to the agent system, an ADP (user client) must be initiated. The ADP must then connect itself to an AMP to establish the connection with the rest of the agent system. This AMP will provide services to the ADP, and thus is called DIAS Service Provider (DSP). An illustration of how an ADP1 connects to AMP2 is shown in figure 15.3.

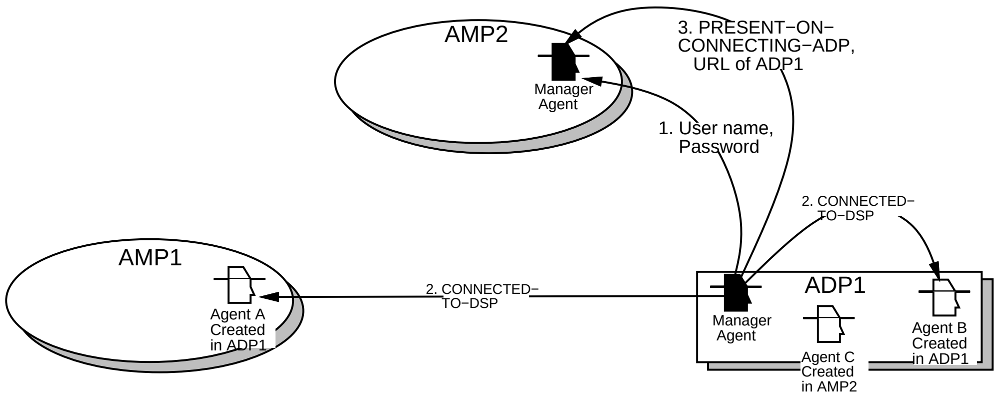


Figure 15.3: Illustration connection between ADP and AMP

The procedure used to handle connection of ADPs, ensures that all parties influenced by the connection are notified. The procedure follows three steps:

1. The connecting ADP1 sends user name and password to the Manager Agent in AMP2 (DSP of ADP1).
2. CONNECTED-TO-DSP tells all agents created in the ADP1 that this ADP is connected to the agent system. This notification tells agents in other agent places (like Agent A in figure 15.3) that it is possible to return home to ADP1.
3. The Manager Agent at the DSP (AMP2) will then receive the message PRESENT-ON-CONNECTING-ADP, along with all needed information from ADP's Manager Agent (the ADP's ID etc.).

When an ADP is disconnecting, the agent system must make sure that all involved parties (agents and agent places) are notified, and status information must be updated. A procedure involving three steps is used when an ADP disconnects:

1. DISCONNECTED-FROM-DSP notifies agents created in the ADP that the ADP is going to disconnect. This makes it possible for these agents to return to home, before the connection with the user is closed down. The user can configure the ADP to wait for a certain time (default five seconds) before it will close down.
2. PRESENT-ON-DISCONNECTING-ADP is sent to all agents that are present on the disconnecting ADP, to notify that they have to move themselves, or they will be stopped.
3. DISCONNECT-FROM-DSP tells the DSP's Manager Agent that the ADP is going to disconnect. This Manager Agent can then delete registered information about the ADP and disconnect it.

15.3.4 Registration of agents and agent places

In DIAS, agent places have registries about agents and other agent places in the system. These registries are essential to make it possible for agents to communicate, and they contain information about ID for agents and agent places. The registration of agents, ADPs and AMPs is done as following respectively:

- **Registration of agents** First, agents will be registered as *origin agent* in the agent place they were created . The origin agent registry hold information about the agent ID, and it is used to inform agents (listed in origin agent registry) that an ADP has moved. A user can choose to start his/her ADP at another site, and the users agents must be informed about current location of the ADP. Second, all agents living in the agent system should be registered as *registered agents* in agent places in the system.

This registry makes it possible to get an overview of agents in the system. If agents also have this agent place as a DIAS Service Provider (DSP), agent location is also registered. Third, agents present at an agent place are registered as *present agents* as well. When the facilitator agent is looking for a receiver of a message, the present agent registry is checked first to see if the message can be delivered locally.

- **Registration of ADPs** A ADP can is only registered in DSPs as “registered ADP”, where general information about the ADP, information about location of the ADP, and ADP’s user information is given.
- **Registration of AMPs** When an AMP wants to register in another AMP, both AMPs exchange information about themselves. In this way the AMPs will have information about each other and be able to recommend each other to agents searching for an AMP hosting a specified agent with given ontology.

15.3.5 Moving Agents

In DIAS, mechanisms for moving agents between hosts are supported in the underlying technology provided by Aglets from IBM. However, the system must also be aware of agents’ location any time. When an agent is moved to another agent place, the agent has to give information to three agent places: (1) To the DSP about where it will move, (2) to the agent place the agent will move from that it has left, and to the agent place the agent will move to that it has arrived. An agent must be registered in the agent place it tries to move to. If not, the agent tries to register itself before moving.

15.3.6 Removing Agents

In DIAS, agents can be created and killed at run-time. However, system agents cannot be killed individually since they must provide their services as long the agent place is running. If the agent places are shut down, the system agents associated with this agent place will also be killed. Participation agents have often a dynamic nature, and can be created and killed on demand. Since these agents are not registered any places, they can be killed immediately. The user agents’ destiny is decided by the user, and it is up to the user to kill them. These agents will unregister themselves from agent places before they are killed.

15.3.7 CORBA Agent Interaction

DIAS offers external programs to access the multi-agent system through CORBA to enable programs in programming languages other than Java or other agent systems to interact with DIAS. In addition, it should be possible to connect to an AMP without having a local ADP installed on the host. A CORBA client can either interact directly with a

CORBA AMP, or interact with a CORBA ADP. The former means that we do not need an ADP installed on the host, while the latter is a practical solution, since both the CORBA programs will be on the same host as the ADP.

The *Interface Agent* plays the main role when CORBA programs interact with DIAS. This agent is created as a system agent in its agent place, and connects to the ORB in the agent place when created. In this way the Interface Agent is an agent as well as being a CORBA object on the ORB. The following interfaces are used for CORBA interaction:

- **MAF AgentSystem interface** Defines agent operations like receive, create, suspend, and terminate.
- **MAF Finder interface** Defines operations for registering and unregistering of agents, and finding the location of agents, places and agent systems.
- **DIAS interface** Defines the communication method used by CORBA applications (can be extended by the developer)

The Interface Agent is the “glue” between the CORBA client and the agents. The operations in the Interface Agent are invoked by the CORBA client. In DIAS, KQML is used to express the communication between agents. When a CORBA client wants to send a KQML message to an agent place (e.g., to initiate an agent) the following will happen:

1. The KQML message is translated to a string.
2. A CORBA call with the KQML string as a parameter is executed.
3. The CORBA object (the Interface Agent) on the CORBA server will then translate the KQML string into a KQML message.
4. The KQML message will then be sent from the Interface Agent to the receiver agent.

15.4 Discussion

The Aglet framework [LO98] provides underlying services for building a mobile multi-agent system, but it lacks of high-level services to enable a fully functional agent system. The design principles in section 15.3 describe how we added these high-level services to provide a better way to manage mobility and distribution of agents, as well as agents interaction. Dealing with mobility and distribution is mainly a technical problem to ensure that the system is reliable and scalable. However, agent interaction, it is not only a technical problem. Our current implementation provides only simple means for agent interaction, and should be replaced with a more advanced mechanism for dealing with this problem. Domain models should be used as ontology to define as limited set of words agents can understand and how these words are related and used. We are currently working on incorporating a document model for modelling ontology. In [GB00], Gulla and Brasethvik offer a framework for creating a graphical model of an ontology and for exporting this model as an XML document. A starting-point for an ontology model is generated using a tool that parse through documents typical for the ontology domain chosen, resulting in a list of domain specific words. A graphical modelling tool can then be used to refine the ontology model and define relationships between words. The last step is to export the ontology model to an XML document that can be used in DIAS.

15.5 Conclusion

In this paper, we have described some design principles applicable when designing a mobile multi-agent system using Aglets technology. The principles are not closely related to the underlying technology, and could therefore also be used for other agent technologies. We found that our design principles will make it easier to generate agent applications, since the agent system takes care of things as routing messages, locating agents, connecting and disconnecting user clients (ADPs). We are now exploring our agent system, building agent applications, to gain more experiences of what advantages and shortcomings our architecture has.

Acknowledgement

We would like to thank Reidar Conradi for useful comments and suggestions. The CAGIS project is sponsored by the Norwegian Research Council's Distributed InformationSystems (DITS) Programme.

CHAPTER 16

Experience paper: Implementing a Multi-Agent Architecture for Cooperative Software Engineering

Alf Inge Wang[1]

Abstract
The paper describes experiences we have earned from implementing a multi-agent architecture used to support cooperative software engineering. Before starting to implement a multi-agent architecture, important decisions and considerations must be taken into account. Decisions on how to provide efficient inter-agent communication support, what language should the agents talk, should the agents be stationary or mobile, and what technology should be used to build the architecture must be made. This paper describes how we implemented our multi-agent system, and the experiences we gained from building it.

Keywords: *Cooperative Software Engineering, Agents, Multi-agent system, KQML, XML, Aglets, JATLite, CORBA.*

16.1 Introduction

The last couple of years, distributed computing and agent technology have become more and more popular. When researchers are developing prototypes, the choice of technolo-

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway. Phone: +47 73 594485, Fax: +47 73 594466, Email: alfw@idi.ntnu.no

gies and how to use different technologies is getting more and more complicated. This paper describes experiences from designing and implementing a Multi-Agent System (MAS) providing support for Cooperative Software Engineering (CSE). CSE is a research area where focus is on how to support participants in cooperative processes working in distributed organisations. In our MAS, agents are used to provide support for cooperative processes as coordination and negotiation. The agents communicate through a defined language, and the MAS offers the infrastructure for the agents interacting with other agents and users.

Before we started to implement a MAS, a theoretical study on distributed architectures and agent technologies was conducted by Joar Øyen, a diploma student at the Department of Computer Science at the Norwegian University of Science and Technology. The outcome of this study was a report [Øye98] giving us guidelines for building a multi-agent architecture (MAA). We have used these guidelines as a starting point for our implementation.

We have been searching for related work that focuses on experiences from implementing MAS using different types of technology. It seams like there is nothing or at least very little published covering these issues. There are a lot of publications on high-level descriptions of MAAs, such as [BJJT99, JLT99, BvET97], but details about technology used or experiences from implementing these systems are left out.

We have used the multi-agent paradigm to implement our system, but there are alternative approaches. JavaSpaces [Mic99b] based on Java RMI and JINI, recently introduced by SUN, provide an alternative approach for exchanging distributed objects. JavaSpaces technology is a unified mechanism for dynamic communication, coordination, and sharing of objects between Java technology-based network resources like clients and servers. A JavaSpace is a virtual space between providers and requesters of network resources or objects. This allows participants in a distributed solution to exchange tasks, requests, and information in the form of Java technology-based objects. There are four primary operations you can invoke on a JavaSpace: **(1) Write** an entry into a JavaSpace, **(2) Read** an entry from a JavaSpace that matches some specified parameters, **(3) Take** an entry from a JavaSpace that matches some specified parameters (removing it), **(4) Notify** a specified object when entries that match some specified parameters are written into this JavaSpace

An entry is in JavaSpace terminology a typed group of objects, expressed in a class for the Java platform. JavaSpace technology offers much of the same functionality as MAS; movement of objects, message handling, sharing of objects, etc. Maybe the most useful functionality in this respect, is the support for searching for objects with certain properties.

16.2 CAGIS Multi-Agent Architecture for Cooperative Software Engineering

This section is a short introduction to the CAGIS[2] multi-agent architecture for Cooperative Software Engineering. A more detailed description of this architecture can be found in [WLC99].

16.2.1 CAGIS Multi-Agent Architecture components

Software agents are useful for supporting cooperative activities, since software agents can act as human agents on behalf of humans. Our architecture uses this property of software agents to model data- and control flow to implement a self-optimising or self-improving process. The main components in this architecture are agents, workspaces, Agent Meeting Place (AMP) and repositories:

- **Agent** A piece of autonomous software created by and acting on behalf of a user. The agent is set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to the environment, as well as pro-activeness. Agents are grouped into three main groups. The first group, *System agents*, is used to execute administrative tasks of the MAA. This can be to monitor human and software agent activity, deal with repositories and managing AMPs (see the third point). The second group, *Local agents*, assist users in work within local workspaces. The last and most important agent group, *Interaction agents*, help users in their cooperative work (coordination, negotiation, communication). Note that mediation agents are also provided to suggest solutions for locked negotiation processes based on prior experiences.
- **Workspace (WS)** A temporary container for relevant data in a suitable format, together with the processing tools. Workspaces can be private as well as shared.
- **Agent Meeting Place (AMP)** AMPs are where software agents meet and interact. AMPs are built on underlying communication mechanisms, but provide agents with more intelligent means to facilitate their interaction. An AMP can also work as a market place where agents can “trade” information and services. The main purpose of the AMP is to facilitate cooperative support for software agents.
- **Repository** A persistent storage of data that can be local, global or distributed. Repositories can be accessed either by tools or by agents. One specific repository is very important in the CAGIS MAA; the experience base. An experience base can be used as the community memory, and a mediator agent can utilise prior stored experiences to solve negotiation conflicts.

Next subsection will give an example of how the CAGIS MAA can be used.

[2]CAGIS (Cooperative Agents in Global Information space) is the name of a Norwegian research project (1998-2000) with main focus on software support for distributed cooperation for human problem solvers. More information about the CAGIS project can be found at:http://www.idi.ntnu.no/∼cagis.

16.2.2 Example of a multi-agent architecture

Figure 16.1 shows how the CAGIS MAA can be used in a simplified software development scenario. This example describes a coordinated software development process involving multiple departments of an organisation. Each department is represented by a workspace[3], and an AMP is used as a place where the departments can cooperate through software agents.

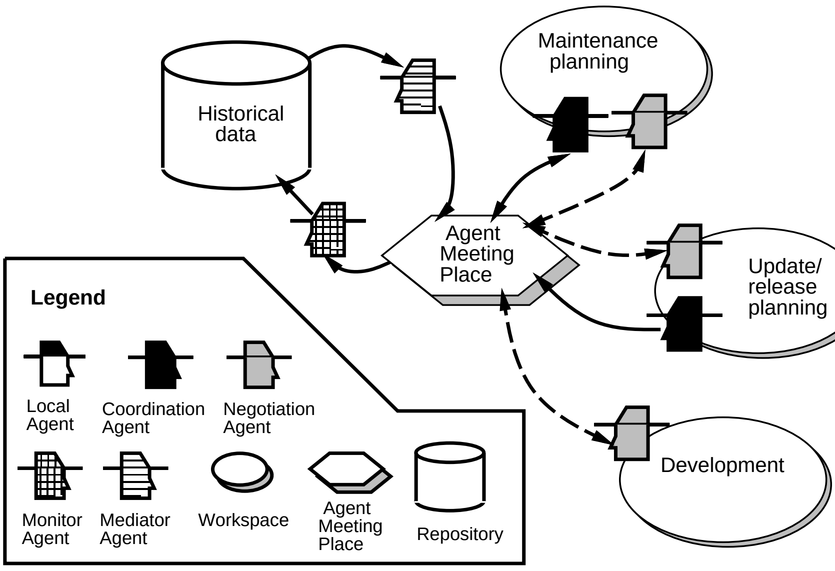


Figure 16.1: An example of an agent architecture application

This scenario demonstrates how it is possible to support a situation where limited human resources in the development department cause trouble for a *Maintenance planning department* and an *Update/release planning department*. Both departments are competing for human resources in the *Development department*, because the two departments want to serve request and complaints from customers, and to improve the software product respectively. If a negotiation process between for instance *Maintenance planning* and *Update/release planning* takes to much time, the mediator agent (shown in figure 16.1) will break into the negotiation process to make an agreement. The mediator agent can base its judgement on experiences from prior projects (e.g., we lost a lost the biggest customer because of too many bugs in the product last year), on company guidelines or through interaction with the company's management.

[3]Workspaces are used in this architecture to group people. It is however possible to have personal workspaces.

16.3 Requirements to the technology used

This section will outline the implementation requirements for the MAA described in section 16.2. These are the requirements for a prototype of the system and not requirements for a full-fledged system. The requirements given in this section should be quite general, and should also be applicable to most MASs supporting mobile agents.

16.3.1 General requirements

The overall goal of the proposed CAGIS MAA was to be flexible and tailorable to many different needs. The main requirement is therefore an open architecture that can evolve together with the real world it is supposed to support.

Since a prototype of the architecture will be built in a short period of time, it is important that the cost of the implementation is kept at a minimum, that is free and proven technologies should be used whenever appropriate. This means that the solution must be feasible today.

Performance is not important for this prototype architecture.

16.3.2 System infrastructure

The organisations that the CAGIS MASs are intended to support, are inherently distributed, a heterogeneous environment, and are continually changing. The infrastructure that is going to be the foundation of the system must thus define an open, flexible, and malleable environment, which encompasses hardware platforms, operating systems and networks.

A run-time environment to implement workspaces and AMPs must be provided by the infrastructure. Such an environment must provide a number of services to its inhabitants. Name registration and service advertising allows agents to be aware about each other's properties. In addition, event mechanisms will allow monitoring agents to register their interest in specific events.

16.3.3 Agent implementation and configuration

Again, an open multi-platform solution is required. This includes the ability to ship binary executables between different platforms, so that the agents can be mobile. Agents are going to be built by experts, but it must be possible to tailor agents to the specific needs of the different users. Some sort of agent scripting or agent configuration is therefore required to let users fine-tune their system.

16.3.4 Agent communication

To be able for agents to communicate, we need a format to represent information, as well as some conversation policies on how agents communicate. The architecture also requires facilities for users to communicate with agents, and agents access to external entities like repositories, workspaces, tools, AMPs etc..

The basic communication mechanisms (streams, messages, events, etc.) are provided by the underlying system infrastructure, and the communication mechanisms uses these services to provide more high-level facilities for inter-agent communication. *First*, agents must be able to communicate with each other in a language that all involved parties understand (not only syntax of the language, but also semantics and pragmatics of the conversation). *Second*, agents must collaborate with each other to reach common goals. This collaboration is controlled by process models that agents must interpret. *Third*, agents also need mechanisms for coordination and negotiation to handle many common situations. Coordination mechanisms can be used to control the concurrency between multiple agents and to exchange instructions that tell agents what to do. The system should provide a negotiation model with a defined context and multiple negotiation strategies. Agents are thus given the possibility to find out what is negotiated for and to select an appropriate strategy to use.

16.3.5 Knowledge sharing

Communication mechanisms like coordination and negotiation operate in a cooperative context and do therefore need to share common goals. The common goals represent some form of group awareness or community memory, and are contained in a work context that consists of information resources and control structures.

Common information can be stored in three different places in the architecture:

1. **In repositories** The main part of knowledge is contained in repositories.
2. **In AMPs** Meta-information about agents are contained in the AMPs, but the actual information is contained inside the agents themselves.
3. **In agents** Agents may be used as information carriers of process information and results.

The most important common requirement is however that information is stored in a formally defined format that makes it accessible by agents.

16.3.6 Design proposal for the prototype

Figure 16.2 shows an overall design proposal for how the prototype should be structured. We have used a multi-tier architecture based on the agent, places, and things paradigm.

The lower part of the figure (component infrastructure and agent infrastructure) will define a foundation, based on available standard implementations, which will provide functionality and services to the prototype of the MAA. In the shown architecture, the facilitators are central components both in workspace and AMP. Facilitators simplify the implementation of agent communication and the coordinator, moderator, and monitor-components. The reason for this is that the interaction between various entities can be controlled from one central point. The drawback with this solution is of course that the facilitators might become bottlenecks of the system.

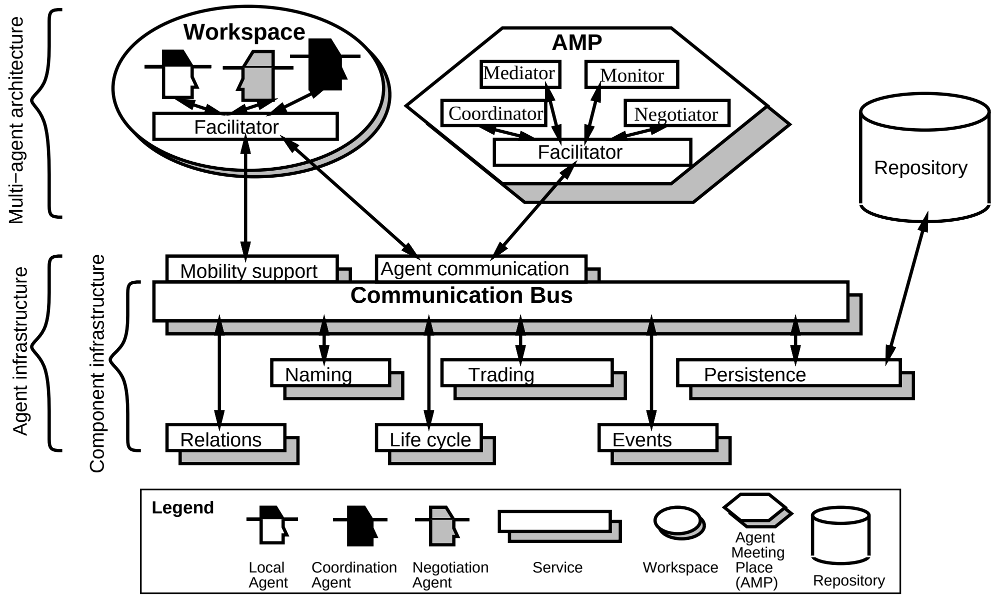


Figure 16.2: Design architecture

16.4 Technological guidelines

This section will give advice on what technology to use to implement a prototype of the MAS proposed in section 16.2 according to the requirements to the technology used described in section 16.3.

1. **Use Java and Java IDL [Mic99c] as the component infrastructure** because (1) Java provides code portability, garbage collection, object-orientation, (2) Java is a broadly accepted standard, (3) Free implementation is available now, (4) Many agent-related technologies are closely associated to Java.

2. **Use the facilitators in the design architecture to implement the trader, event, lifecycle, and relationship services as needed**. By choosing Java IDL for the

component infrastructure, we have to use this CORBA implementation for the rest of the system as well. This gives us a major drawback, because Java IDL only provides the naming service of the CORBA services shown in figure 16.2. Since the design shown in figure 16.2 has several services that are not supported in Java IDL, these services must be developed in Java if required. These services can be replaces, if Java IDL will offer them in the future.

3. **Use Aglets to implement the mobility support for the agents**. Mobility support should ideally be implemented by OMG's Mobile Agent Facility, but due to the fact that this standard is under construction, Aglets will be recommended instead [4]. Aglets [Lan97, OKO98, LA97] are Java objects, developed at IBM's Research Laboratory that can move from one host on a computer network to another. Aglets servers provide distribution of aglets to aglets viewers. The viewers are the users graphical interface to the aglets. From this interface the users can create, activate, dispatch, and retract agents.

   Using Aglets should provide an easy transition to the Mobile Agent Facility later, because Aglets are one technology that is used as a basis for the Mobile Agent Facility. Another advantage with the Aglets-technology is that user-interfaces for controlling the operation of the agents are provided as a part of the technology.

4. **Use KQML and JATLite for agent communication and as a foundation for implementing workspaces and AMP**. The Java Agent Template Lite (JATLite)[JPC00] is a technology that provides a Java implementation that lets agents communicate with each other, possibly by using Knowledge Query and Manipulation Language (KQML). We recommend using KQML [FLM97] as the agent communication language, because it is an extensible standard that has many of the advantageous features that an agent communication language should have. JATLite is also a technology that can be used to implement workspaces and AMPs, because each JATLite-router defines an environment that facilitate the administration of and communication between a group of related agents.

5. **Use XML to represent information and work-products in the architecture**. eXtensible Markup Language (XML)[Hol98] does not put any restrictions on the format of the information is shall represent. Also tools to support XML are available in Java.

How the recommended technologies are used to implement the various parts of the design architecture is illustrated in 16.3. It must be noted that figure 16.3 only loosely denoted where the various technologies should be used, and more experience is needed to decide exactly which of the technologies that are best in the specific situations.

[4]At the time these technological guidelines was worked out, OMG's standard for Mobile Agent Facility was not finished, and there was not any implementation of OMG's Mobile Agent Facility.

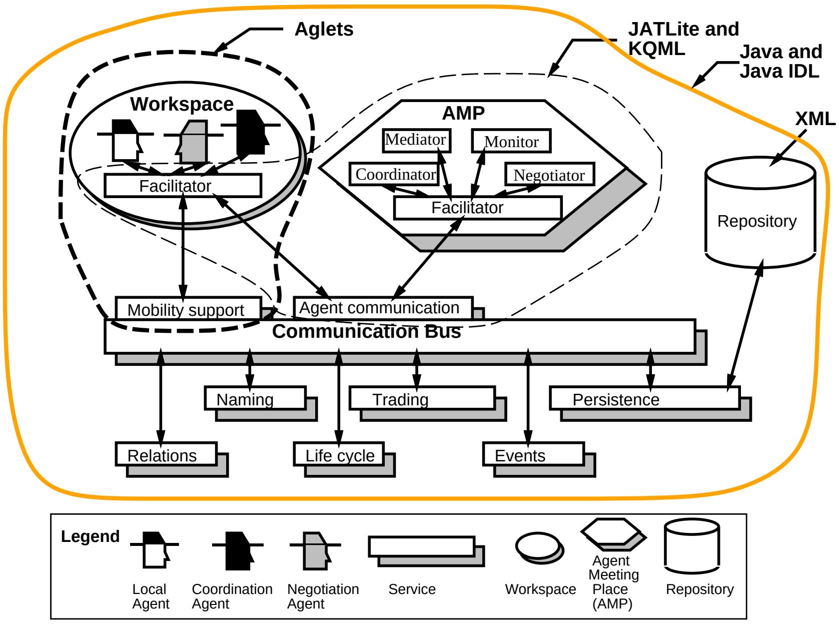


Figure 16.3: Implementation of the design architecture

16.5 Experiences from implementing a MAS based on the guidelines

The implementation of our MAA is named Distributed Intelligent Agent System (DIAS). We have implemented two versions DIAS namely DIAS I and DIAS II. Both implementations were developed by last year students at the Dept. of Computer and Information Science at the Norwegian University of Science and Technology. This section describes their work and experiences.

16.5.1 DIAS I

Spring 1999, four students used about 1000 hours to make the first version of the implementation of our MAA[PHBN99]. These students were skilled in Java-programming and XML. However, the area of programming agents and implementing a MAS was totally new to them. DIAS I used the requirements to technology (described in section 16.3) and technological guidelines (described in section 16.4) as a starting-point for the prototype.

Choice of technology

We discovered that not everything from section 16.3 and 16.4 could be used directly when implementing the system. The first problem we encountered was how to combine JATLite and Aglets into one implementation. JATLite was well suited to implement facilitators for workspaces, AMPs and generally for agent communication. Unfortunately, JATLite does not support mobile agents. Since we wanted to have support for mobile agents, we had to find a way of making JATLite mobile. The Aglets framework was suggested to be used to support mobility, but then we had to integrate JATLite with Aglets. We found that this was not an easy task. We chose to use Aglets as the main implementing standard for agents and agent communication, and just to use parts of the KQML layer in JATLite. This solution makes it possible to use mobile agents in all parts of the architecture. It was possible to use parts of JATLite because JATLite is open-source. The parts missing in Aglets compared to JATLite was quite easy to implement using the functionality found in Aglets. Another reason for using Aglets was that the documentation was better than JATLite.

The requirements to technology and technology guidelines suggested that we should use CORBA (Java IDL) as the communication bus in our architecture. When we worked with the first DIAS-project, the mobility support over the ORB was not possible because the standardisation Mobile Agent System Interoperability Facility (MASIF) was not approved yet by the OMG. Since we wanted to keep the mobility support for agents, Aglets was chosen for communication over TCP/IP. Aglets was chosen because Aglets was one of the agent languages that MASIF is based on. This should make the transition to MASIF easier on a later stage.

KQML was chosen as communication language for our architecture, since KQML is the most used standard for agent communication. Parts of the KQML layer in JATLite were used to give the architecture KQML support. Since the designer is free to choose the representation format of the content in a KQML message, we decided to use XML for this purpose. XML is a good choice when the knowledge is going to be stored in a web-server or in a file-system. XML has also become a well know standard for information representation.

Experiences of use

Although the first version of DIAS was not very advanced, the architecture was sufficient to demonstrate parts of the scenario described in section 16.2.2. Negotiation agents for the groups *Maintenance planning* and *Update/release planning* used to negotiate about limited human resources were implemented. The experiment showed that such negotiation processes were supported in our architecture through negotiation agents and an AMP. The agents were able to move between workspaces and the AMP, and they were able to communicate using KQML as a communication language. The following KQML performatives were used in our agent-language (these performatives were sufficient at least to demonstrate our scenario):

- **ask-if** Sender wants to know if the sentence is in the Virtual Knowledge Base (VKB) of the receiver.
- **insert** The sender asks the receiver to add content to its VKB.
- **register** The sender can deliver performatives to some named agents.
- **tell** The sentence is in the VKG of the sender.
- **unregister** A deny of a register performative.
- **untell** The sentence is not in the VKG of the sender.

We found some shortcomings with the DIAS I implementation. **First**, it was rather hard to implement user agents, because substantial knowledge of the Aglets framework was required. The low-level agent API made it hard to experiment for with own user agents. **Second**, the implementation was lacking of high-level support for inter-agent communication and inter-AMP communication. The latter meant that to support several AMPs, the programmer of the agents had to hard-code how to deal with the different AMPs and agents. This was not desirable. **Third**, the DIAS I implementation did not provide support for integrating the MAS with other systems.

Because of the shortcomings described above, we decided to continue with a DIAS II project (see next section)

16.5.2 DIAS II

After finishing implementing DIAS I, two students continued the DIAS project as diploma thesis's, and spent about 1000 hours to extend the original implementation to DIAS II [HN00]. The DIAS II project focused on making the original DIAS implementation better. The following problems were addressed: agent security, integration with other systems through CORBA, and making a higher-level agent API.

Choice of technology

In the first version of the DIAS implementation (developed in Java JDK 1.1), agent security was not addressed at all. In JDK 1.2, the enhanced security model for fine-grained resource access has been added. Because of this, JDK 1.2 would be a natural choice for DIAS II. Unfortunately, the Aglet Software Development Kit, ASDK 1.1 does not support JDK 1.2. This meant that we had to abandon the new security features in JDK 1.2 if we would like to keep Aglets to support mobility. By comparison, we found that ASDK actually had almost the same security features as in JDK 1.2 included, so we chose to use JDK 1.1.

Since we wanted to add possibility to integrate our MAS with other systems, CORBA support was needed for our architecture. One way of giving our architecture CORBA support was to change Aglets with other mobile agent implementations supporting CORBA. Both

Odyssey [Whi96] and Voyager [Gla99] supports CORBA had provide interesting functionality for distributed systems and interoperability between different communication facilities. Another alternative was to continue using Aglets and to implement CORBA support into it. We chose to do this, because it required less work and the Aglets implementation have better support for security. OrbixWeb [Bak97] was used as a CORBA implementation, because of its functionality and availability at the University. External systems can now communicate with our MASs in the AMPs with CORBA support. The CORBA support is implemented according to OMG's Mobile Agent System Interoperability Facility (MASIF). MASIF is a standard to make it possible for interoperability between various multi agent systems, and have four areas that are standardised: Agent management, agent transfer, agent and agent system names, and agent system type and location syntax. The rest of the implementation of DIAS II uses the same technology as in DIAS I.

Experiences of use

The agent-API for DIAS II is totally different compared to the first version of DIAS. For the first version, you had to write low-level Aglets-code to make an agent. In DIAS II, you don't have to know that the architectures use Aglets technology at all. The following areas are covered with high-level methods in the agent API:

- **Create/kill agent** These methods are called when creating new agents or when you want to kill an agent. Agents will be moved back to where they were created before it will be killed. Note that agents can also be cloned.
- **Message handling** Methods for sending/receiving KQML messages to/from other agents. The architecture will take care of sending messages to the correct receiver. The sender will always receive an acknowledgement when the receiver has received the message.
- **Register/unregister agents** These methods are used to register/unregister agents in AMPs.
- **Move agents** Methods for moving an agent to another AMP. If an agent cannot find what he is looking for in an AMP, the AMP can suggest the agent to move to another AMP.
- **Information queries** Various methods offer the agents the possibility to ask AMPs for information about what agents are connected, what type of ontology is used, what properties have other agents etc.

Another major change of the architecture is how agents communicate, and how AMPs can communicate. In DIAS I it was necessary to explicitly state how the communication between agents should be performed. In DIAS II, the system takes care of looking for agents, using advanced communication agents. Agents are located according to agent ID number, AMPs ID number or/and ontology of AMPs. If for instance neither the agent ID number or AMPs ID number is know, the ontology is used to find a matching agent.

16.6 Conclusion

Implementing a MAS is not an easy task. The technologies available, makes it easier to build robust systems in rather short time. It is however important to know what technologies are available and what to choose before starting building a MAS. It is also important to know what technologies are possible to combine, before starting on the work. Our experience with building a multi-agent system is that the standards available should be used when possible. Using OMG's MASIF standard will make it possible for a system to communicate with other MASs as well to other applications through CORBA. KQML is the most widely used agent communication language used, and makes it possible for agents on different systems to communicate. XML offers a convenient information wrapping that is useful for different purposes in an MAS, as information/knowledge representation, small repositories etc. Using Java as the programming language, makes it possible for the system to run in an heterogeneous environment, and most agent standard implementations as well as XML tools are implemented in Java as well.

We are now using our CAGIS MAA for cooperative software engineering to make support for various scenarios. In doing this, we want to see how general our architecture is and recognise the shortcomings of our architecture.

Acknowledgement

We will give a big thank to Joar Øyen who has provided us with in-depth information about distributed technology in theory and in practise. We would also like to thank Geir Prestegård, Snorre Brandstadmoen, Anders Aas Hanssen, and Bård Smidsrød Nymoen for implementing DIAS I. Anders Aas Hanssen and Bård Smidsrød Nymoen were also responsible for the DIAS II implementation.

CHAPTER 17

Support for Mobile Software Processes in CAGIS

Alf Inge Wang [1]

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

17.1 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

[1]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

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 [MLL97, STO99].

In 1986-1996, our research group, managed by Reidar Conradi, worked on a PCE prototype called EPOS [CHLN94]. 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) [pro00] 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) [WLC99]. 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 [WLC99], we presented our CAGIS multi-agent architecture for cooperative software engineering (see figure 17.1). 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 [LO98] are used to support mobile agents [PHBN99]. In figure 17.1, 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.

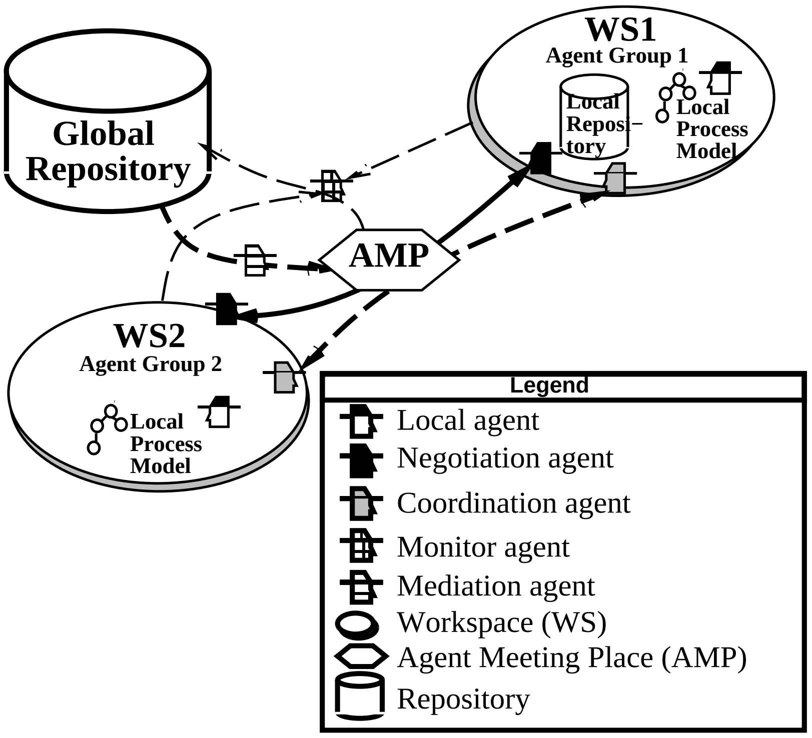


Figure 17.1: MAS-based architecture for cooperative software engineering

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 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 17.2 present related work on distributed PCEs, and mobile workflow. Section 17.3 presents the main concepts of our approach. Section 17.4 briefly outlines some preliminary experiences, and discusses problems and possible solution for our prototype. Section 17.5 concludes the paper.

17.2 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 [Tia98], 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 [BSK95], 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 [BCN+96], 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 [BO96], 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 [YL99], 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 [JHS$^+$99], 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 [Tia98], Ben-shaul [BSK95], and Basile et al. [BCN$^+$96]) discuss how to support federation of PSEE. Our paper does not discuss federation in particular, but touches issues such as local autonomy and distribution of process fragments. None of these papers look into how to support mobile software processes. Bhattacharyya and Osterweil [BO96] however, describe how to analyse the impact of relocation of a process fragment. We only describe the mechanisms to provide mobile process fragments, and does not provide advanced tools for analysing the impact. The two last papers described in this section, discusses mobile workflow from two different perspectives. Yoo and Lee [YL99] provide mobile workflow by making the process instances mobile agents, while Jing et al. [JHS$^+$99] 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.

17.3 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.

17.3.1 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 [Yeo96]). 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[2]. 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 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 17.2. 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 17.3.2). 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.

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 17.2, can be a process fragment.

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

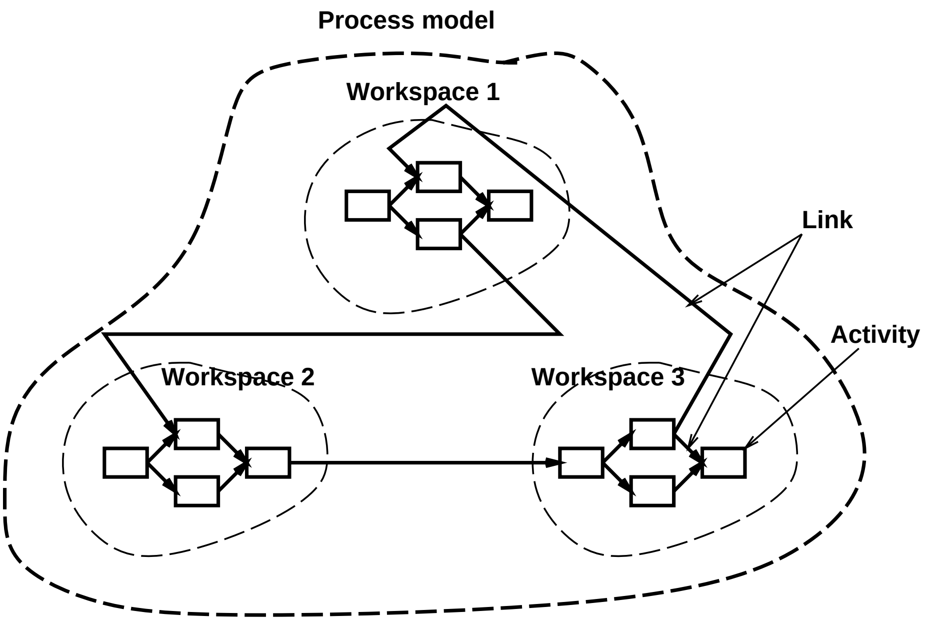


Figure 17.2: Composition of a process

But a process fragment can also be any combination of related activities as shown in figure 17.3. 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) [Wan99] 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 17.3.2.

17.3.2 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 17.4, 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 17.4) a process consists of one or more process fragments, and

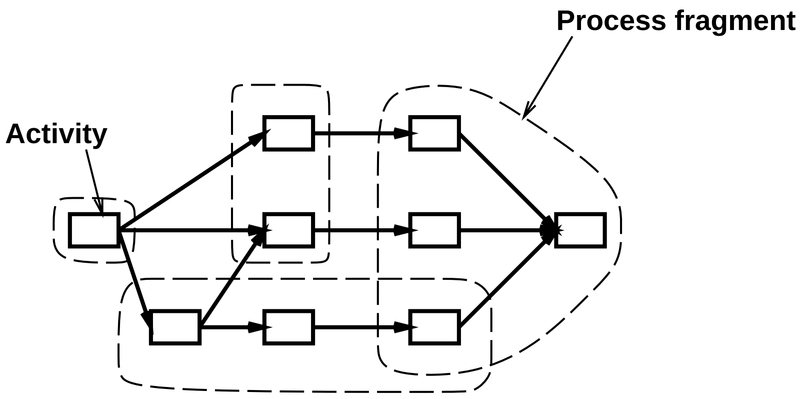


Figure 17.3: The concept of process fragment

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. 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 17.3.4). 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 17.5 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.

Figure 17.6 shows how the dependencies between the activities are specified in our PML (XML syntax) for the example shown in figure 17.5. 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.

```
<?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)>
```

Figure 17.4: XML Document Type Declaration of our PML

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 17.3.4) 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.

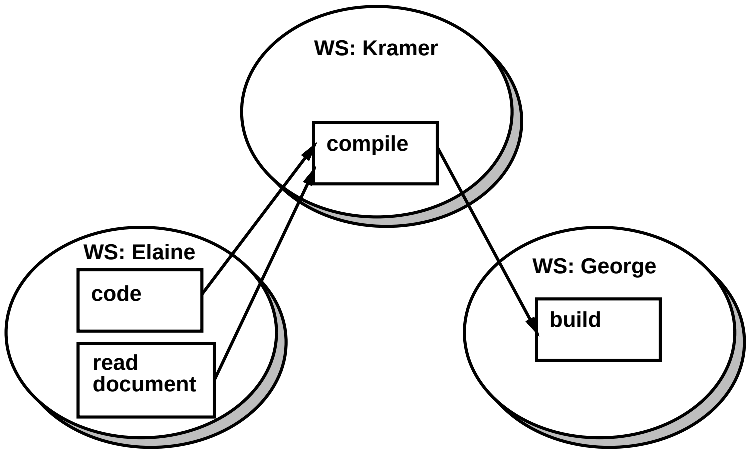


Figure 17.5: Example with linked activities in several workspaces

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

Figure 17.6: An example of use of $< prelink >$ and $< postlink >$ tags

17.3.3 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 17.7. 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.

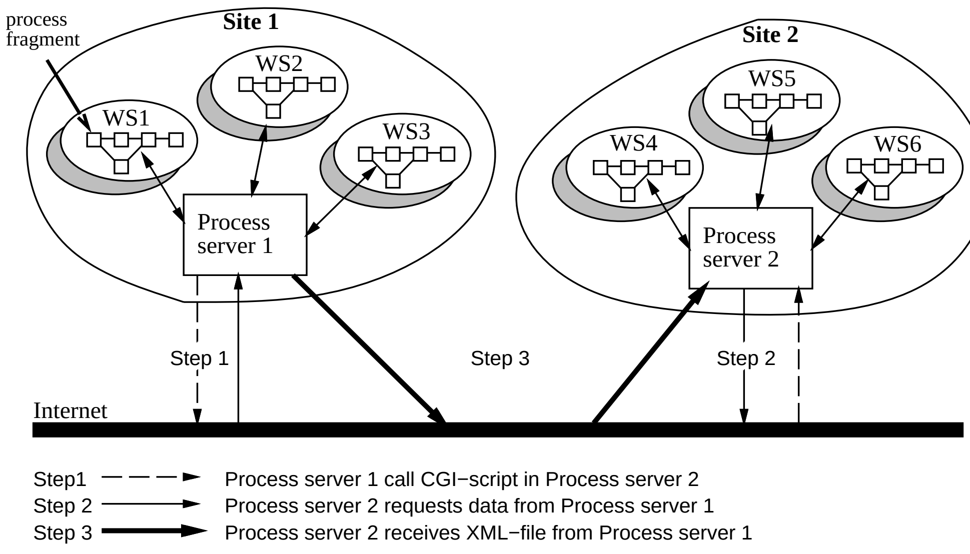

process fragment
Site 1
WS1
WS2
WS3
Process server 1
Site 2
WS4
WS5
WS6
Process server 2
Step 1
Step 3
Step 2
Internet
Step1 Process server 1 call CGI−script in Process server 2
Step 2 Process server 2 requests data from Process server 1
Step 3 Process server 2 receives XML−file from Process server 1

Figure 17.7: Moving process fragments between different process servers

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 engineering described briefly in section 17.1 can be used for this purpose (see also [WLC99, PHBN99] 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 [BHT97] is one example of such a system.

17.3.4 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:

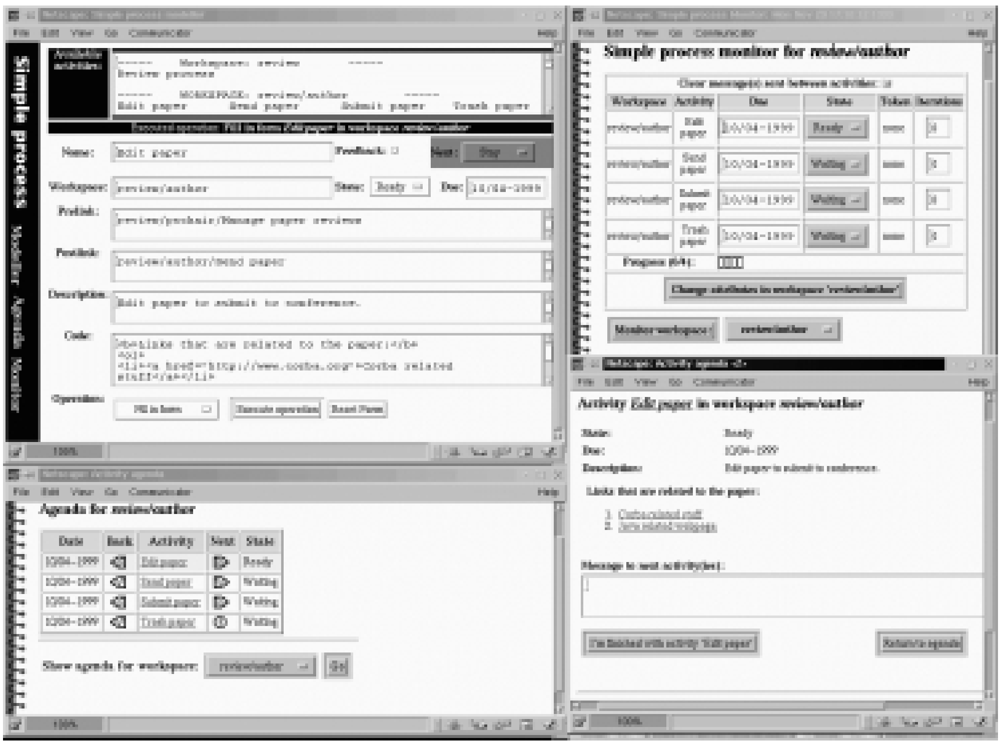
- **Process server/engine** The process server has three main tasks. *First*, it manages the state information of activities. Through CGI, the state of an activity can be changed. Synchronisation of an activity having several prelinks, is done by checking if all prelinks have the state *Finished* before 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 [WCL00]. 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 17.8, 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



```
------    Workspace: review       ------
Review process

------    WORKSPACE: review/author       ------
Edit paper      Send paper      Submit paper    Trash paper
```

Executed operation: Fill in form *Edit paper* in workspace review/author

Name:	Edit paper
Feedback:	☐
Next:	Stop
Workspace:	review/author
State:	Ready
Due:	10/04-1999
Prelink:	review/pcchair/Manage paper reviews
Postlink:	review/author/Send paper
Description:	Edit paper to submit to conference.
Operation:	Fill in form

Code:

```
<b>Links that are related to the paper:</b>
<ol>
<li><a href="http://www.corba.org">Corba related
stuff</a></li>
```

Simple process monitor for review/author

Clear message(s) sent between activities: ☐

Workspace	Activity	Due	State	Token	Iterations
review/author	Edit paper	10/04-1999	Ready	none	0
review/author	Send paper	10/04-1999	Waiting	none	0
review/author	Submit paper	10/04-1999	Waiting	none	0
review/author	Trash paper	10/04-1999	Waiting	none	0

Progress (0/4):

Change attributes in workspace 'review/author'

Monitor workspace: review/author

Agenda for review/author

Date	Back	Activity	Next	State
10/04-1999	◁	Edit paper	▷	Ready
10/04-1999	◁	Send paper	▷	Waiting
10/04-1999	◁	Submit paper	▷	Waiting
10/04-1999	◁	Trash paper	⊙	Waiting

Show agenda for workspace: review/author Go

Activity *Edit paper* in workspace review/author

State:	Ready
Due:	10/04-1999
Description:	Edit paper to submit to conference.

Links that are related to the paper:

1. Corba related stuff
2. Java related webpages

Message to next activity(ies):

I'm finished with activity 'Edit paper' Return to agenda

Figure 17.8: Screenshots of the Process modeller tool, the Agenda manager tool, and the Monitor tool

left, the agenda for the workspace *review/author* is shown. To the right, the activity *Edit paper* is shown.

17.4 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 [Yeo96]. 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 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 17.1), 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 [WLC99], 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.

17.5 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.

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 on how to improve the paper.

CHAPTER 18

Integrating Software Process Fragments with Interacting Agents

Alf Inge Wang, Reidar Conradi[1], and **Chunnian Liu**[2]

Abstract
Cooperative software engineering processes involve structured, repeatable processes as well as dynamic, cooperative processes. Existing workflow systems are suited to model and support the former type of processes, and multi-agent systems are suited to model and support the latter. We have designed and implemented a gluing-framework for integrating workflow processes with software agents. By using this framework, support for cooperative software engineering processes can be provided in a better and more flexible way. This paper focuses on how to integrate these two kinds of components into a functioning multi-agent based cooperative software engineering system.

Keywords: Agents, Cooperative software engineering, Workflow, Web.

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technolog (NTNU), N-7491 Trondheim, Norway, +47 73 594485
alfw/conradi@idi.ntnu.no

[2]Beijing Polytechnic University (BPU), Beijing, P.R. China, bpvliu@public.bta.net.cn. Chunnian Liu's work was supported in part by the Natural Science Foundation of China (NSFC), Beijing Municipal Natural Science Foundation (BMNSF), and the 863 High-Tech Program of China.

18.1 Introduction

Cooperative Software Engineering (CSE) means that large-scale, software development and maintenance can be conducted in a distributed organisation or across organisations. CSE can be regarded as a special case of Computer Supported Cooperative Work (CSCW). If we categorise CSCW according to increasing complexity of process support [LC98], CSE falls into the most complex category of CSCW, characterised by distributed process fragments, partly shared workspaces, cooperation planning, and frequent interactions in intra/inter-workspaces. To support CSE processes, you have the problem of dealing with dynamic, unpredictable processes as well as stable, repeatable processes with totally different characteristics. Workflow and process systems like InConcert [Sar96], Lotus Notes [Orl92], Active Mail [GSSS92], and MAFIA [LvRH90] offer good support for stable, pre-planned processes, providing users activity agendas, invocation of tools, presentation of the state of the process etc. Multi-agent systems are better fit to model and support users involved cooperative processes [MM97, WJ95, DFJN97]. By combining these two technologies, cooperative software engineering processes can be modelled and supported more completely.

In [WLC99], we proposed a multi-agent system (MAS) architecture for CSE, which is an extension and specialisation of the more general architecture [MDP98] for CSCW. The MAS is particularly useful in modelling and providing support to cooperative activities (communication, coordination, collaboration and negotiation). We have also developed two prototyped systems based on the proposed architecture. One is a multi-agent architecture [PHBN99, HN00, SW00, Wan00a] supporting cooperative work through software agents. The other, is a workflow tool [Wan99, Wan00b] to model and support simple stable, repeatable processes. In this paper we describe how we combined these two systems, by using a GlueModel defining interaction between these systems, and a GlueServer executing the GlueModel and communicating with the agent system and the workflow tool. By doing this, we integrate the workflow process with interacting agents into a MAS-based CSE system.

There are several advantages by combining workflow technology with multi-agent technology. Dynamic cooperative processes between people are hard to model in traditional process modelling languages, because these languages have difficulty representing interaction between autonomous participants. However, *software agents* provide a natural way of modelling dynamic cooperative processes by letting a software agent play a role to represent a person. In this way, the software agent can execute, e.g. a negotiation process on behalf of a person and his wishes. Software agents are also useful when there is an uncertainty in the process regarding resources, availability, the state of work etc. Software agents can be used to gather required information for making decisions on what to do next in the process. For more structured and more repeated processes, workflow tools can provide a better support. This is because they make the process modelling simpler by offering process modelling languages specialised for modelling such processes. Also many companies have workflow tools already installed, and want to keep them. By combining these technologies, we can benefit from both.

Our approach provides a framework for doing *component-based workflow*. This means that process fragments (parts of the whole process model) can be combined in a flexible way. The GlueModel defines the rules for how the workflow components can be combined and (re-)configured, and software agents are used to get information from the working environment to make the correct decisions. The process fragments are highly distributed, so process owners can change and manage their process fragments themselves. The GlueServer framework can also be applied on other workflow systems, by using the Interoperability Workflow-XML Binding framework. Since GlueModel can be used by different workflow systems, an organisation can let sub-organisations use different types of workflow tools locally. The GlueModel will define the interaction between these sub-organisations, and interactive-agents will execute the according cooperative processes.

18.2 Related work

In [CS96], Chang and Scott present an agent-based workflow system. They claim, that unlike groupware products that simply provide a passive information space, agent-based workflow can enable active collaborative work among participants. The agent-based workflow system proposed in their paper support people working in different places, to work together at the same time in virtual rooms. The core of the system is based on WWW and a set of distributed agents to facilitate various parts of workflows (as activity agendas, document sharing and tool sharing) to improve efficiency among participants. The architecture is based on stationary agents, and the HTTP-protocol serves as an infrastructure for agent-communication. The coupling to workflow is provided through a workflow agent that interfaces with an existing workflow repository through a standard Workflow Management Coalition API. This means that various workflow tools can access the workflow repository, while the workflow agents act as facilitators, providing an interface between other agents and the workflow tool.

In two papers [STO99, MLL97], Shepherdson and Merz et al., present two approaches to use software agents for cross-organisational workflow. Both papers claim that an obstacle for cross-organisational workflow is that organisations have to give up their local autonomy to cooperating partners. One workflow system for the cooperating partners is therefore not an appropriate choice. In [MLL97], Merz et al., propose to use mobile agents to overcome problems for cross-organisational workflow, typically lack of a common communication infrastructure, lack of central management, and high coordination costs of workflow management systems. Their multi-agent infrastructure serves as a communication mechanism that bridges organisational boundaries. In [STO99], a approach called *agent enhanced workflow* has been chosen. Agent enhanced workflow is achieved by combining a layer of agents with a commercial workflow system. The agent layer is given responsibility of business process management. The advantage of this approach is to provide automatic provisioning, interoperability, support for visualisation and verification services, while protecting the original investment in workflow technology.

The main difference in our approach compared to the approaches described above, is that we use agents only to model and support cooperative activities between actors, while a traditional workflow system is used to model and support isolated activities. We propose a model and a framework to glue these two different models and systems, to benefit from both.

18.3 Review of MAS-based architecture for CSE

In our MAS-based CSE architecture, there are four components [WLC99]:

1. *Agents*: An agent is a piece of software created by and acting on behalf of the user or some other agent. It is set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to environment, as well as pro-activeness. We distinguish between the following different types of agents: *Work agents* to assist in normal software production activities, *Interacting agents* to help participants in their cooperative work, and *System agents* to give system support to other agents.

2. *WorkSpaces (WS)* A workspace is primarily a temporary container for relevant data in a suitable format, together with the processing tools. Workspaces can be private or shared, and the work in a workspace is often guided by a process model where data can be checked in and out from persistent repositories.

3. *AgentMeetingPlaces (AMPs)* AgentMeetingPlaces are where agents meet and interact. AMPs are built on the underlying communication mechanisms, but provide agents with more intelligent means to facilitate their interaction by using KQML [FFMM97] as a communication language and a defined ontology.

4. *Repositories* Our architecture allows persistent repositories to be global, local or distributed. Repositories can be used as a storage for products, but also as an Experience Base.

In the architecture, the four components are interconnected and interoperated as follows:

1. Agents are created by people to assist their work; or by other agents to perform delegated work; or by default to manage WSs or AMPs. In this paper we are concerned mainly with interacting agents triggered through the GlueServer by a workflow tool.

2. Agents are clustered mainly according to people grouping in workspaces.

3. Communication between agents is via AMPs.

4. Within a group of agents and their shared WS, any existing process models and their tools are allowed, and any traditional process architecture can be applied. Different process fragments may be expressed in different Process Modelling Languages (PMLs) and require different process tools.

Figure 18.1 shows an example of our MAS-based architecture for CSE containing workspaces, one AMP, agents and repositories. The *negotiation agents* and *coordination agents* are used to provide cooperative support between the workspace WS1 and WS2.

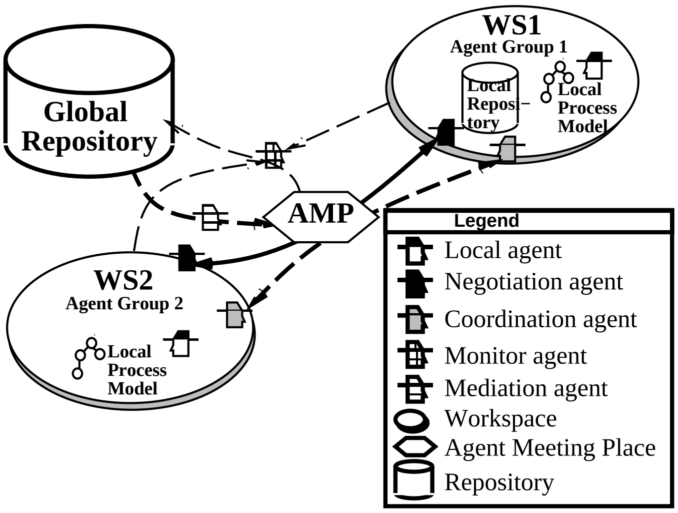


Figure 18.1: An example of the CAGIS MAS-based architecture for CSE

18.4 Gluing workflow with interactive agents

In a distributed setting, the overall process of a software development and maintenance project is distributed on various WSs, so the local software process running in a particular WS is a fragment of the overall process. Our main focus in this paper is the integration of process fragments with interacting agents. A process fragment can be seen as a sub-process of the overall process, and the Workflow Management Coalition defines a sub-process as the following: *”A process that is enacted or called from another (initiating) process (or sub process), and which forms part of the overall (initiating) process. Multiple levels of sub process may be supported.*” [WfM99]

The smallest building block in our workflow PML is an *activity*. We define a *process fragment* as a part of a process consisting of partial ordered activities, located in a workspace, and its *working context* (data, tools, human roles and agents). Our framework defines the integration between software process and interacting agents in terms of process fragments, rather than the whole process or individual activities. We are not loosing any generality by doing this, because a fragment may consist of a single activity. A process fragment is identified by a name, workspace (URL), a list of activities and associated data. Since the workspace is identified by a URL, the process fragment name and workspace name make an unique identifier.

18.4.1 The GlueServer

The main functionality of the GlueServer is communication between workflow systems and the multi-agent system, and installation and execution of the GlueModel. The GlueServer is implemented in Java using OrbixWeb CORBA implementation, and it consists of four main parts (see figure 18.3):

1. **FragmentServer** The FragmentServer receives requests from the workflow system. A *GlueServer request* formatted in XML looks as shown in figure 18.2. It describes the identifier for the process fragment requesting the GlueServer, the other process fragments involved (interactors), and the name of the agent-class that should be initiated.

```
<GlueServerRequest>
  <From>Fragment-id</From>
  <Interactor>Fragment-id</Interactor>
  ...
  <AgentClass>Agent-Class</AgentClass>
</GlueServerRequest>
```

Figure 18.2: Workflow request to GlueServer

2. **GlueEngine** The GlueEngine parses the GlueModel, and finds matching fragment-agent pairs based on agent-class and fragment-ids (more on this in section 18.4.2).
3. **AgentClient** When fragment-agent matches are found in the GlueModel, the Agent-Client connects to the multi-agent system and initiate agents as specified in the GlueServer request and the GlueModel.
4. **FragmentClient** Depending on the result returned from the initiated agents and what is specified in the GlueModel, the FragmentClient will send a response back (formatted in XML) to the workflow system on what actions to take.

fragment-agent connection

We perceive the following types interactions between a workflow system and interacting agents:

- *Predefined Interface*: Some process fragments have a predefined interface with interacting agents. For example, a fragment may have a negotiation activity that is delegated to an agent. When the execution arrives at that activity, a negotiation

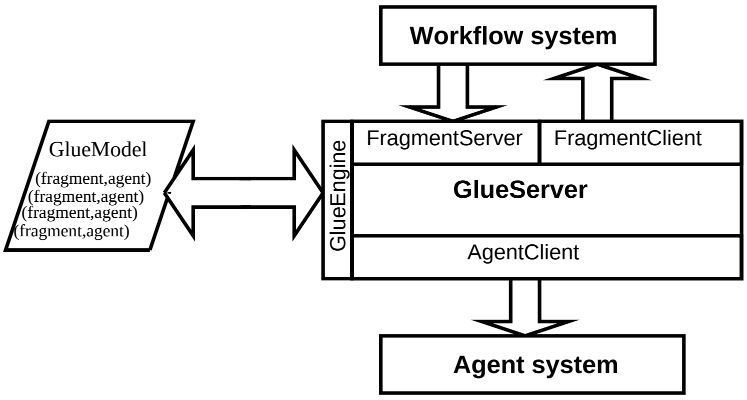


Figure 18.3: GlueServer architecture

agent should be initiated, and the fragment may wait until the negotiation result is reported by the agent.

- *Periodic Invocation*: For example, by the termination of a fragment, usually a decision will be made about the next step of the process. The possible decisions include: Going on to the next fragment, iterating on the current fragment, changing the model of current fragment and re-executing it, moving the fragment to another WS, etc. The decision may depend on environmental information, reported by interacting agents.

- *Dynamic Monitoring*: Some monitoring agents may be designed to continuously probe the events and status of the environment. Whenever an abnormal situation is detected by an agent, the monitoring agent should report the information to relevant process fragments and the latter should be aware of the information in time and react accordingly.

18.4.2 The GlueModel

We use the GlueModel to specify how process fragments and interactive agents interact. The GlueModel is specified in XML, and process fragments and interactive agents are grouped into *(fragment, agent)* pairs. These pairs model all possible inter-operations of fragments and agents in a CSE system.

Result-reaction pairs

The formalism specifies both the agent and the process fragment. The agent specification contains the class of the involved agent, the identifier of the AMP where the agent works, the type of interaction (communication, coordination, negotiation, etc.), and the result reported by the agents. The fragment specification contains the process fragment identifier, a list of *(result, reaction)*-pairs specifying the corresponding reaction by the process fragment for each possible result reported by the agent. Figure 18.4 shows the definition of the GlueModel given in XML document type definition. The set of possible reactions are shown in figure 18.4 as the attribute *body* in the XML-element *action*.

```
<?xml version="1.0" encoding="UTF8"?>
<!ELEMENT GlueModel (fragment-agent-pair)+>
<!ELEMENT fragment-agent-pair (agent,fragment)>

<!ELEMENT agent (interaction-type,result)+>
<!ATTRLIST agent agent-class CDATA #REQUIRED
                 amp-id CDATA #IMPLIED>
<!ELEMENT interaction-type (#PCDATA)>
<!ELEMENT result (#PCDATA|result)*>

<!ELEMENT fragment (reaction)*>
<!ATTRLIST fragment fragment-id CDATA #REQUIRED>
<!ELEMENT reaction (result,action+)+>
<!ELEMENT action EMPTY>
<!ATTRLIST action fragment-id CDATA #IMPLIED
 agent-id CDATA #IMPLIED
 workspace-id CDATA #IMPLIED
 amp-id CDATA #IMPLIED
 body (execute_process_fragment_PFNUMBER
 |move_process_fragment_PFNUMBER_to_workspace_WSNUMBER
 |halt_process_fragment_PFNUMBER
 |change_and_reexecute_process_fragment_PFNUMBER
 |add_new_process_fragment_PFNUMBER
 |remove_process_fragment_PFNUMBER
 |start_new_interaction_by_AGENTNUMBER
 |stop_interaction_by_AGENTNUMBER
 |create_a_new_AgentMeetingPlace_AMPNUMBER
 |remove_AgentMeetingPlace_AMPNUMBER
 |change_fragment-agent-pair_PFNUMBER_AGENTNUMBER)
#REQUIRED>
```

Figure 18.4: Fragment-agent-pair specification

In the above formalism, a keyword marked by '+' or '*' means "one or more" or "none or more", respectively. The PFNUMBER, AGENTNUMBER, WSNUMBER and AMPNUMBER are used as place-holders for identifiers to process fragments, agents, workspaces, and agent meeting places respectively. We have implemented a graphical Java-based tool to make it easier to enter fragment-agent pairs. A screenshot of the GlueModel Maker is shown in figure 18.5.

18.4.3 Interaction with other workflow tools

Current implementation of the GlueServer is not capable of interaction with other workflow systems than our own prototype. We will however describe how other workflow systems can be integrated in the system using the Workflow Management Coalition Interoperability workflow-XML binding specification (Wf-XML) [WfM00]. The intention of this specification is to allow workflow systems supporting simple chained and nested

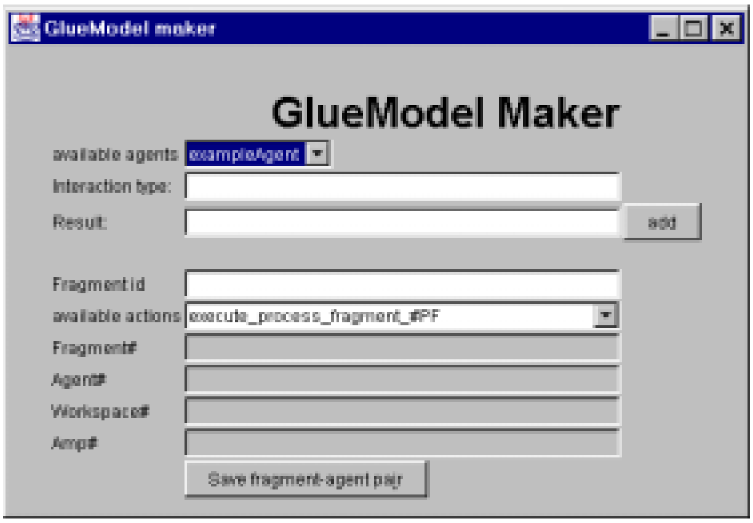


Figure 18.5: GUI tool for adding fragment-agent pairs to the GlueModel

workflow interoperate both asynchronously and synchronously. The Wf-XML defines two types of messages that can be sent between workflow systems; requests (initiate an operation on a remote resource), and response (send the result of an operation to its requesting resource). In addition four operations are defined for Wf-XML messages; CreateProcessInstance, GetProcessInstanceData, ChangeProcessInstanceState, and ProcessInstanceChanged. These four operations could typically be used as reactions in the GlueModel. Note that the reactions described in section 18.4.2 are not supported by the operations defined in Wf-XML. Figure 18.6 shows how a GlueServer request from a Wf-XML compatible workflow system would look like.

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
  <WfMessageHeader>
    <Request ResponseRequred="yes"/>
    <Key>{Interactor process fragment}</Key>
  </WfMessageHeader>
  <WfMessageBody>
    <ContextData>
      <AgentClass>{Agent class}</AgentClass>
    </ContextData>
  </WfMessageBody>
<WfMessage>
```

Figure 18.6: GlueServer request written as a Wf-XML request

18.5 Application scenarios

In [WLC99], we presented a scenario based on the software development and maintenance process from a Norwegian software company, in this paper called AcmeSoft. In this section, we will look at some cases of interconnection and interoperation between process fragments and interacting agents, and show how the GlueServer improves support for these situations.

First we recapitulate the relevant parts of the scenario. In company AcmeSoft, among other things, there is a Maintenance Planning Group (MPG), a Release Planning Group (RPG), and a Development Group (DG). The MPG major concern is to plan and estimate error corrections to existing software products based on error reports from customers. The MPG process consists of the following process fragments: *register_report, estimate_resources, allocate_resources, sends_out* (a Change Order to the DG) etc. On the other hand, the RPG plans what new functionality should be implemented in existing software products based on requirements from emerging technology and market demands. AcmeSoft is committed to quarterly update-releases and one major release per year per product. The RPG process also includes fragments like *estimate_resources*, *allocate_resources* and *sends_out* (a Work Order to the DF). Finally, all real development and correction work is carried out by the DG consisting of software engineers. The process fragments *estimate_resources* in MPG/RPG try to allocate a particular developer (or a group of developers) from DG for a particular maintenance/develop task. The identity of a fragment is determined by a fragment name and a WS name. For example, the process fragment *estimate_resources* in MPG will be denoted as MPG/*estimate_resources*. To give unique identifiers to workspaces, they should be named with an URL (like http://www.acmesoft.no/project/MPG). In this paper we name the workspaces only with the local workspace name.

18.5.1 Scenario 1: Negotiation

The first situation we consider, is when MPG and RPG compete for a particular developer. Remember that the same DG is responsible for both maintaining existing products, and for developing new updates/releases. Suppose that both MPG and RPG are trying to allocate the same developer because of his/her special skills. Here we have a conflict in resource allocation between the process fragments estimate_resources for MPG and RPG. To solve the conflict, negotiation between MPG and RPG is necessary. In this case, two negotiation agents representing the two groups (equipped with problem-specific negotiation strategies) carry out the inter-group interaction. The negotiation process will be carried out in Acme Agent Meeting Place (AMP), providing inter-agent communication facilities. Figure 18.7 shows the XML for the fragment-agent pairs involved in the negotiation process.

This situation shows how periodic invocation (see section 18.4.1) can be used to combine workflow with agents. When MPG and RPG are estimating resources, they both will

```
<fragment-agent-pair>
  <agent agent-class="agents.Negotiation" amp-id="AcmeAMP">
    <interaction-type>negotiation</interaction-type>
    <result>yes|no</result>
  </agent>
  <fragment fragment-id="RPG/estimate_resources">
    <reaction>
      <result>yes</result>
      <action fragment-id="RPG/allocate_resources"
      body="execute_process_fragment_PFNUMBER"></action>
        <result>no</result>
        <action fragment-id="RPG/allocate_resources"
        body="change_and_reexecute_process_fragment_PFNUMBER">
        </action>
    </reaction>
  </fragment>
  </fragment-agent-pair>
  <fragment-agent-pair>
    <agent agent-class="agents.Negotiation" amp-id="AcmeAMP">
      <interaction-type>negotiation</interaction-type>
      <result>yes|no</result>
    </agent>
  <fragment fragment-id="MPG/estimate_resources">
    <reaction>
      <result>yes</result>
      <action fragment-id="MPG/allocate_resources"
              body="execute_process_fragment_PFNUMBER"></action>
        <result>no</result>
        <action fragment-id="MPG/allocate_resources"
                body="change_and_reexecute_process_fragment_PFNUMBER">
      </action>
    </reaction>
  </fragment>
</fragment-agent-pair>
```

Figure 18.7: GlueModel for scenario 1

access a shared repository denoting what resources they will need. As a result, the workflow tool will be able to know if a resource conflict has occurred. After the termination of for instance MPG's process fragment *estimate_resources*, the workflow tool will send a FragmentServer request for negotiation between MPG and RPG where the agent-class is specified. The GlueServer will then look in the GlueModel for matching fragment-agent pairs both for MPG and RPG, and initiate the agents involved in the negotiation process. The result of the negotiation process will be sent back to the GlueServer, and the GlueServer will forward the reactions, according to the GlueModel, back to the workflow tools in MPG and RPG.

18.5.2 Scenario 2: Coordination

Suppose that a particular developer in DG is assigned to a maintenance task by MPG, but quits or is given another job before the task can be completed. Assume that DG is not sure if they can assign this job to another of their own developers, or if they give the assignment to an external organisation. Then the DG (or rather, its process fragment *report*) would report this difficult problem to MPG. The *replanning* process fragment in MPG may decide to delegate the unfinished work to someone else in DG if possible, or an external organisation (company or another department). Coordination agents are used to decide whether the work should be re-assigned in DG or to an external organisation. The coordination agents will check updated information about the workload of DG. If the outcome of the coordination is to assign the task to an external organisation, it will move the remaining work together with its process fragment PF# to the external work space WS#. A GlueModel for scenario 2 is shown in figure 18.8.

```
<fragment-agent-pair>
  <agent agent-class="agents.Coordination" amp-id="ExternalAMP">
    <interaction-type>coordination</interaction-type>
    <result>local|external</result>
  </agent>
  <fragment fragment-id="DG/report">
    <reaction>
      <result>local</result>
      <action fragment-id="DG/implement_changes"
              body="change_and_reexecute_process_fragment_PFNUMBER">
      </action>
      <result>external</result>
      <action fragment-id="DG/implement_changes"
              body="halt_process_fragment_PFNUMBER"></action>
    </reaction>
  </fragment>
</fragment-agent-pair>
<fragment-agent-pair>
  <agent agent-class="agents.Coordination" amp-id="ExternalAMP">
    <interaction-type>coordination</interaction-type>
    <result>local|external</result>
  </agent>
  <fragment fragment-id="MPG/replanning">
    <reaction>
      <result>local</result>
      <action fragment-id="MPG/estimate_resources"
              body="change_and_reexecute_process_fragment_PFNUMBER">
      </action>
      <result>external</result>
      <action fragment-id="MPG/estimate_resources"
              body="change_and_reexecute_process_fragment_PFNUMBER">
      </action>
      <action fragment-id="DG/implement_changes" workspace-id="EXTERNAL"
             body="move_process_fragment_PFNUMBER_to_workspace_WSNUMBER">
      </action>
    </reaction>
  </fragment>
</fragment-agent-pair>
```

Figure 18.8: GlueModel for scenario 2

In scenario 2, MPG will issue a request to the GlueServer to initiate a coordination process involving the process fragments MPG/replanning and DG/report. Coordination agents will find a solution to who is going to finish the job specified in the process fragment *implement_changes*. Depending on the result of the coordination (local/external), *implement_changes* will be executed by DG or an external organisation. For the fragment-agent pair (MPG/replanning, agent.Coordination), the result *external* will cause more than one action to be executed. Note also that the coordination process is executed in ExternalAMP. This is to make the coordination possible beyond local organisational borders.

18.5.3 Discussion

As we have shown in the two scenarios above, our framework for combining process fragments with interactive agents enable us to model and support dynamic and ad-hoc cooperative processes with software agents, while traditional workflow process described in some process modelling language takes care of the routine processes. In scenario 1, the software agents are used to negotiate about human resources between two groups. The GlueModel is used to define how the workflow tools shall act according to the result of the negotiation between the software agents. This means that each group can model the local process concerning themselves on their own, while the GlueModel will model how to deal with interaction with other groups. Scenario 2 shows how the glue model can deal with exception handling, when a particular developer assigned for a specific task is resigning or leaves. In this scenario, we could also use software agents to search for external human resources capable of doing this specific task. In addition, the software agents could form a market-place where they can look for expertise and negotiate about the price to do a specific job. The glue model will then specify what activities to do next based on the result returned by the agents.

The GlueModel is a part of the total process model, and defines how different cooperating groups shall interact. Agents are then used to do inter-group interaction as representatives for the groups. It is also possible to use the GlueModel to specify for what situations local process models have to be changed, based on work-environment informations collected by agents. This means that the GlueModel also models part of the meta-process. The GlueModel will notify the local workspace, when a process model must be changed, and the changes will be executed locally in the workspace. In addition, the GlueModel is reflective. This means that a reaction in a fragment-agent pair can cause a change in the same or other fragment-agent pairs.

When the GlueServer architecture is applied to a specific environment, one or more GlueServers can be used. For small GlueModels (less than 50 fragment-agent pairs), one GlueServer can be used. If larger GlueModels are demanded, several GlueServers should be used. By using several GlueServers, it will be easier to maintain the GlueModels. This is because each GlueModel will be smaller, and it is more likely that the maintenance of the GlueModel will be done by a person that has more local knowledge of how people cooperate for a given group. In this way, area of concern for maintaining the GlueModel can be distributed according to how people are organised. Using several GlueServers will also shorten the GlueServer response time. Since both AMPs and GlueServers provide cooperative support between workspaces, one GlueServer per AMP could typically be used. A GlueServer should be maintained by higher management, for instance a project manager should be responsible for a GlueServer serving a project team. However, all people affected by a GlueModel must participate in order to create a useful GlueModel. The GlueModel will evolve over time, and hopefully reflect the cooperation protocols between actors involved.

The GlueServer was relatively easy to implement, and it runs on Java Virtual Machine, with OrbixWeb and Sun's XML package installed. Agent-interaction is provided through OMG's MASIF standard, making it possible for the GlueEngine to interact with other agents systems. Results sent back from the agent system is translated from KQML to XML in the GlueServer. Multiple workflow interfaces can be added to the GlueServer through CORBA, CGI-calls or Java RMI.

18.6 Conclusion

We have presented a framework for giving flexible process support to cooperative software engineering processes, combining software agents with traditional workflow systems. The framework consists of a GlueModel specifying process-agent interaction and a GlueServer providing integration of agent systems with workflow systems. Some real-world CSE scenarios suggest that the mechanism is conceptually concise, and practically useful. We are currently working on a validation of the GlueServer framework, where other real-life CSE scenarios are modelled using other existing systems as well. From this work we would gain more experiences in the advantageouses and disadvantageouses our framework offers.

Acknowledgement

We want to thank to Bjørn Haakenstad for implementing the GlueServer, and executing the scenarios. The CAGIS project is sponsored by the Norwegian Research Council's Distributed Information Systems (DITS) Programme.

CHAPTER 19

Using software agents to support evolution of distributed workflow models

Alf Inge Wang[1]

Abstract
This paper outlines a high-level design of how software agents can be used combined with an existing CAGIS Process Centred Environment to deal with evolution of distributed, fragmented workflow models. Our process centred environment allows process fragments of the same workflow model to be located in workspaces that are geographically distributed. These process fragments can be changed independently in local workspaces causing consistency problems. We propose to use software mobile agents, offering awareness services solving conflicting updates of process fragment. Our solution is illustrated using some scenarios.

Keywords: Process centred environments, software agents, workflow model consistency, workflow model evolution, distribution, fragmentation.

[1]Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway, Email: alfw@idi.ntnu.no, Phone: +47 73594485

19.1 Introduction

Dealing with evolution of workflow processes is not a trivial matter. One simple solution to this problem is to have one centralised workflow model, that cannot be changed after it is instanciated. In practice, it is however hard to follow a process that cannot change. Most processes have uncertainties, and it is therefore impossible to model everything correct in advanced. In our CAGIS Process Centred Environment we have proposed a workflow tool where the workflow model is distributed into smaller parts called *process fragments*, that can individually be changed locally. This gives the users the opportunity for local adoptions, and therefore make the workflow model representation closer to the real process. All this freedom can also cause problems with keeping the workflow model consistent. Typical process changes can be re-arranging the order for when activities should be executed, adding new activities, removing or merge activities, changing the contents (code) of a activity, re-allocating activities, and re-scheduling activities.

In [NWC97], Nguyen et al. presents results from studying process changes in development projects in a Norwegian banking software house. These process changes are causing late project deliveries and bad resource and cost estimates. The four most typical reasons for changing the development process were found to be: **(1) Customer postponent** The customer requests unexpected postponing of the start date, causing a forward adjustment of the entire project plan/schedule. **(2) Customer delay** The customer delays with delivering documents (e.g. requirements), causing re-allocation of already scheduled activities and forthcoming dependent activities. **(3) Customer misunderstanding** The customer misunderstands or ignores important details in the initial requirement specification causing backward adjustment or rework of completed activities. **(4) Customer revision** The customer issues new or revised requirements due to better insight to the problem domain causing forward adjustment by introducing additional activities to incorporate new or revised features.

In [WLCM98b], we present work on how cooperation support can be improved in a configuration management system dealing with consistency problems when changing documents. This problem is quite similar to dealing with consistency of workflow models, but the latter is a more complicated matter. Workflow models define how people work in an organisation, and how people are supposed to cooperate. Changes of workflow models would therefore in most cases impact on how people are organised. These aspects are not so relevant document consistency are in focus.

Most work on maintaining consistency in workflow and process systems have been focusing on database and transaction support for these systems. In [Cla93] describing Coo, Godart proposes to use long transactions to save intermediate results, and that several software processes can access these intermediate results without violating the correctness criterion for the transaction. Intermediate results are managed as three different consistency levels; *stable*, *semi-stable*, and *unstable*. TransCoop/CoAct [Rol98] is another work in this research area where the motivation was to overcome the limitations imposed by the use of a standard ACID model. The requirements for the transaction model were defined by using four application scenarios: Cooperative authoring (ad-hoc processes), design for manufacturing (structured activities), software engineering (semi-structured processes), and workflow (automated business processes). CoAct uses advanced transaction models, and operations are exchanged between workspaces instead of exchange of data as in Coo.

19.2 The CAGIS Process Centred Environment

In 1997, a project called *Cooperative Agents in Global Information Space* (CAGIS) [pro00] was started. One of the main goals of the CAGIS project was to see how heterogeneous, distributed work could be supported. As an outcome of this project, we have implemented a process centred environment (PCE) prototype to give process support to cooperative software engineering (CSE) processes. The CAGIS PCE consists of three main components:

- **Workflow System supporting Distributed Mobile Processes** This workflow system is used to model simple, repeatable workflow processes, and the system offers agenda-browser for the end-users. The workflow system allows an instanciated workflow model to be distributed as several process fragments on different workspaces. One benefit of allowing several instances of a workflow model to be distributed and fragmented over several workspaces, is the possibility to adapt the workflow to local environmental conditions. The workflow instances are defined as xml-files located in the local workspaces, and can can be changed at any time. The ability to move workflow instances during enactment, can be used for reallocation of activities, dealing with exceptions (someone responsible for a particular activity is sick), and delegation of work. For a more detailed description, see [Wan99, Wan00b].

  The Process Modelling Language (PML) for the workflow system defines a process as set of activities that can have pre-order relationships between them specified in XML syntax. An *activity* can specify a set of *pre-links* identifying what activities to be executed before, and *post-links* identifying activities to be executed after the activity this particular activity. The pre- and post-links can be written as URLs, and allow therefore the process to be distributed over several workspaces. Every activity definition specify a code part. This code part is simply HTML, and can be used to simple present text, to specify a form, or to start a Java-applet. The term *process fragment* is used to name a group of activities in a workspace, that is one part of the whole process. A process fragment is specified by a name, a workspace (location), and a list of references to activities.

- **Software Agents to support Dynamic, Cooperative Processes** While the workflow system described above takes care of simple, repeatable process, we use software agents to support more cooperative and dynamic processes. Software agents typically takes care of inter-workspace activities as negotiation activities (e.g., about of resource allocation), coordination of artifacts and workflow elements between workspaces, brain-storming, voting, marked support (in a multi-company scenario, we can perceive that agents acts as buyer and sellers of services), etc. Our multi-agent architecture consists of four main elements:

- **Agents** An agents is set up to achieve a modest goal, characterised by autonomy, interaction, reactivity to environment, as well as pro-activeness. We have identified three main types of agents: (1) *Work agents* to assist in local production activities, (2) *Interaction agents* to assist with cooperative work between workspaces, and (3) *System agents* to give system support to other agents. Interaction agents are mobile, while system and work agents are stationary.
- **Workspaces** A workspace is a temporary container for relevant data in a suitable format to be accessed by tools, together with the processing (work) tools. It can be private, as well as shared. Files stored in a repository can be checked in and out to a workspace.
- **Agent Meeting Place (AMP)** AMPs are where agents meet and interact. AMPs provide agents support for doing efficient inter-agent communication. There can be different types of AMPs for different purposes. Each AMP will have a defined ontology, that the agents have to follow. We can perceive special AMPs for negotiation, coordination, information exchange, selling and buying services etc.
- **Repositories** Repositories can be global, local, or distributed, and are persistent storage of data. Experience Bases are one specific type of repository we can use in our multi-agent architecture to support community memory.

More detailed description of the multi-agent architecture can be found in [WLC99, PHBN99, HN00].

- **Agent-Workflow Glue Server** The Agent-Workflow Glue Server facilitates means to specify how the workflow system and the multi-agent system shall interact. A **glue model** defines the relationship between workflow elements and software agents. The Glue Server will therefore provide support, so that a workflow activity trigger an agent and vice versa. More information about the Glue Server can be found in [WCL00, Bjø00].

Figure 19.1 shows a simplified illustration of how the different components in the CAGIS PCE interact. In Figure 19.1, there are two workspaces, each running a workflow tool with a local workflow model. In reality, this workflow tool can be shared, and the local workflow models in the two different workspaces can have relationships between them. The figure illustrates two different ways that software agents can interact with workspaces. In the first way, the agents can interact directly with the user in the workspaces, using a graphical user interface to configure and interact with the agents. In the second way, the user does not interact with the software agents directly. All interaction with software agents goes through the Glue Server and the workflow tool. The workflow tool can activate an agent, or an agent can activate the workflow tool. The figure also shows that agents can be used to access repositories, but workspaces can also access files in the repository directly (not shown explicitly in the figure).

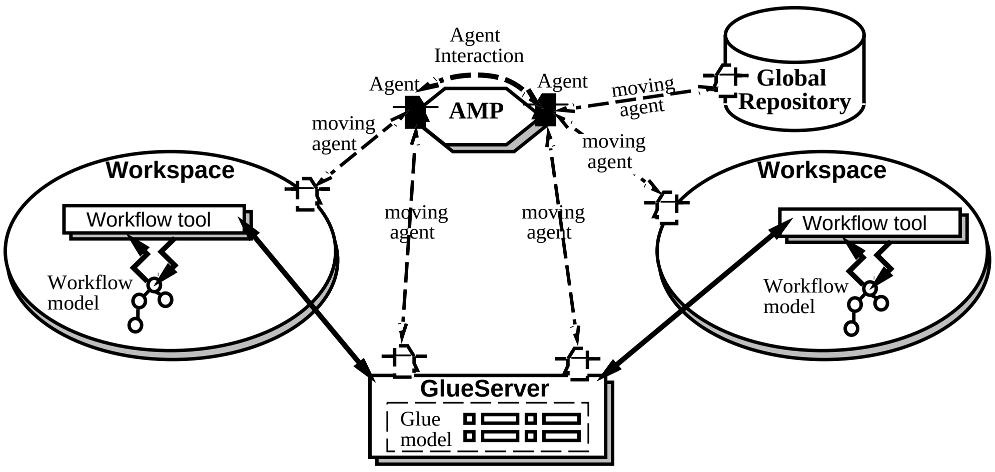


Figure 19.1: The CAGIS Process Centred Environment

19.3 The Problem scenarios

The current version of our CAGIS PCE has no restricting for users to evolve workflow models during enactment. This means that every user can in principle change his/her definition of the workflow process in his/her workspace. Since the definition of the workflow model is specified in XML (also instances of the workflow model), you can simply change the workflow model using a text-editor. In our CAGIS PCE, a workflow model can consist several process fragments that can be updated and distributed separately. This means that a workflow model can be distributed over several workspaces as process fragments, and each process fragment can be changed locally. By allowing this, we give freedom for the every user of the workflow system to adopt their local process to their daily practise. Typically local adoption could be to, add new activities, re-arrange the order for when activities are going to be executed, to change code (HTML) of an activity an activity (make it more close to reality), make a better estimate of time-consumption etc. By allowing these local changes without putting any restrictions it will be very hard to ensure consistency for the whole workflow model. For instance to re-arrange the execution-order of activities in one workspace, could cause problems for activities in other workspaces having pre/post-order links to these activities. Also if multiple instances of the same activity get changed differently in various workspaces, it will be hard to know what version is the correct one.

In the rest of this section we will look at some scenarios that identifies different workflow access policies for an organisation as shown in Figure 19.2. The organisation is responsible for executing various research experiments, and is has a very simple structure consisting of one manager, two project managers for project 1 and project 2, and three project members in each project.

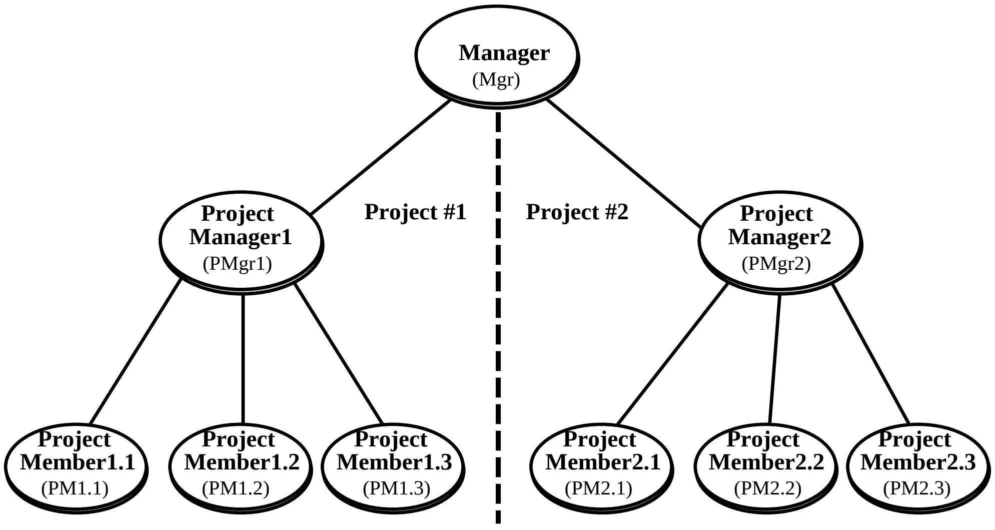


Figure 19.2: Organisational hierarchy used in the scenarios

Note that the shape of the organisation is created to make it easier to illustrate the different scenarios.

19.3.1 Scenario 1: Anarchy

The *Manager* wants all the project members in project 1 and 2 to do the same experiment in parallel. This means that the *Manager* distributes the process fragment **Parallel-Experiment** to the project managers *Project Manager1* and *Project Manager2*. *Project Manager1* is not quite happy with the workflow definition of how **Parallel-Experiment** should be executed, and makes changes to it (makes a new version **Parallel-Experiment** *version PMgr1*). The project managers distribute their two different versions of **Parallel-Experiment** to their respective project members. Some of the project members (both in project 1 and 2) are not still happy with how the process fragment is defined, and creates their own versions of it.

In this scenario, we have a very chaotic situation with a lot of different versions of what should be copies of the same process fragment. As a result the experiment will be executed in different manners, and it will be hard to know what is the correct version to be used. Even worse, if activities for one project member have been dependent on other activities for another project member, it would be likely that changes in one workspace would corrupt the whole workflow model.

19.3.2 Scenario 2: Exclusive update on one process fragment

As for scenario in section 19.3.1, the *Manager* wants to do the same experiment, but this time the process fragment **Parallel-Experiment** can only be updated in one workspace at a time. When this project starts, only the *Manager* can make changes to the process fragment. If for instance *Project Manager1* want to make changes to the process fragment, the *Manager* must give the permit to change it to *Project Manager1*. In this fashion, the process fragment can only be changed in one workspace at a time.

Although, this scenario is not as chaotic as last scenario, there are also problems to be solved for this scenario. What happens if the process fragment **Parallel-Experiment** is changed during the execution of this project ? How should the changes be propagated ?

19.3.3 Scenario 3: Exclusive update on related process fragments

In this scenario, the three project members in project 1 and 2 are responsible of different part of the experiment. This means that the the process fragments for the three project members will not be executed in parallel, but each project member has activities that are dependent on the other project members activities. The process fragment **Intervened-Experiment**, consists therefor of three process fragments that are interrelated. Since it is only allowed for one workspace to change a group of process fragments that have interrelationships, only the project managers and the *Manager* can make changes. Typical dependencies between process fragments are here pre-order relationships between activities.

Here we can get problems if *Project Manager1* makes changes to the process fragment **Intervened-Experiment**, and *Project Manager2* does not. Should two different versions of the process fragment be allowed during enactment ? What version to be used for later similar projects ?

19.3.4 Scenario 4: Exclusive access

The *Manager* wants to execute a one-person experiment defined in the process fragment **Solo-Experiment**. This process fragment is distributed to *Project Manager2*. *Project Manager 2* executes this process fragment using the workflow system. After doing the **Solo-Experiment**, *Project Manager2* recognise that the process fragment has to be changed to adopt some environmental conditions, and it is then distributed to *Project Member2.2*.

In this scenario, only one workspace can access a process fragment at a time. This means that only one workspace can read and/or update a process fragment simultaneously. This cooperative protocol ensures the consistency of the process fragment, but also puts a lot of limitations for how it can be used. It is not a problem for how to propagate changes, since there is always only one instance of the process fragment around.

19.3.5 Scenario 5: Level-based access

The *Project Manager1* has decided that his three project members only have exclusive read/update access (see section 19.3.4) to the process fragment **Solo-Experiment**. This means that only one project member can access (read and/or update) the process fragment simultaneously. One level up in the organisational hierarchy, the project managers have decided to have exclusive update access (see section 19.3.2) for the process fragment **Solo-Experiment**. This means that both *Project Manager1* and *Project Manager2* can get read access to process fragment **Solo-Experiment**, but only one of them can update it. *Project Manager2* has decided that his project members can do what-ever they want with the process fragment (anarchy, see section 19.3.1).

This scenario illustrates that it is possible to have different access policies for different groups in the organisation. There is however one important restriction for how different access policies can be used: A workspace can not have a access policy that violate its parent workspace's access policy. We can identify four levels of access policies according to strictness (1 is most strict, 4 is least strict):

1. Exclusive access (section 19.3.4)
2. Exclusive update on related process fragments (section 19.3.3)
3. Exclusive update on one process fragment (section 19.3.2)
4. Anarchy (section 19.3.1)

If for instance, *Manager* at the top in the organisation hierarchy, has decided to have exclusive access on a specific process fragment, the rest of the organisation need to do the same. Note that the project members in project 2 can only use access policy 4, if*Project Manager2* has update access. A problem working in this manner is if one part of the organisation wants to change their access policies, this can also cause changes on higher and lower levels in the organisation.

19.4 Awareness Agents dealing with Workflow Evolution

From the scenarios in section 19.3, we have identified four access policies that should be supported in the CAGIS PCE. The consistency problems in the three least strict access policies (2-4) must be addressed and solved. To solve consistency problems, we need a *workspace manager* to take care of versioning of process fragments, as well as access restrictions and *awareness services*. In Figure 19.3, an overall design of the workspace manager is illustrated.

The figure shows how the workspace manager interact with the repository and different workspaces. In the repository, workflow templates and models are stored, as well as model of how the workspaces are organised. Nested transactions are supported through the workspace manager and the repository. Process fragments can be moved between

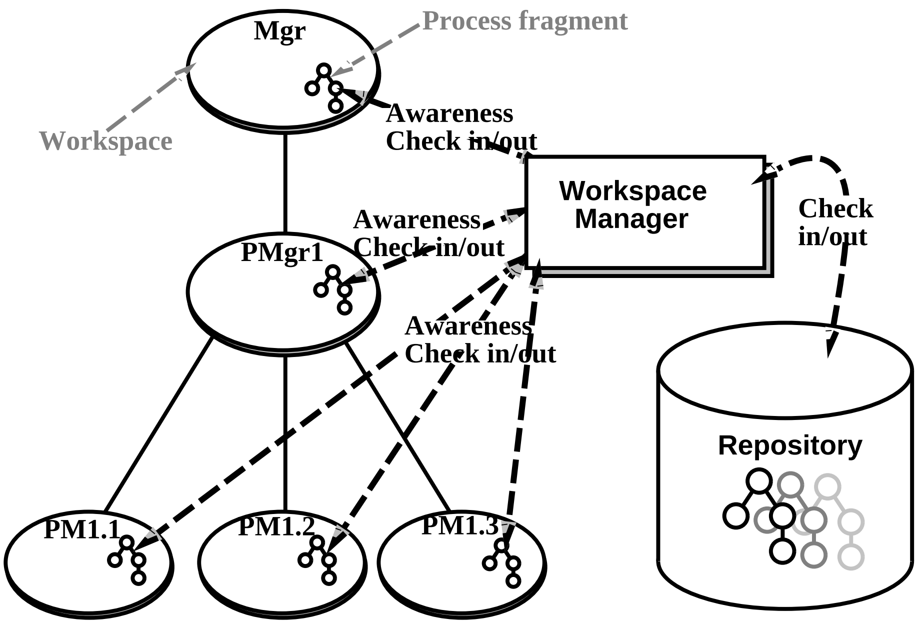


Figure 19.3: Illustration of the workspace manager

workspaces, managed by the workspace manager. In addition, we need awareness support to deal with consistency problems. We suggest to use software agents to provide awareness services, and to resolve model consistency problems.

The rest of this section describes how the support for the five scenarios described in section 19.3 is supported by the workspace manager and software agents.

19.4.1 Support for scenario 1

In scenario 1, there is no access restrictions for a specified process fragments. All workspaces can read and change this process fragment as much they want. As the workspace manager supports nested transactions, consistency can be obtained by solving consistency problems on one level in the workspace hierarchy at a time (bottom-up). When e.g., in the workspace PM1.1 checks out the process fragment **Parallel-Experiment** for updating, and the workspaces PM1.2 and PM1.3 have checked out the same fragment for read, nothing happens. If however, PM1.2 also checks out **Parallel-Experiment** for updating, there is a conflict between the workspaces PM1.1 and PM1.2. The workspace manager will then activate the awareness service by sending two *negotiation agents* to the respective workspaces. The negotiation agents will notify the users in workspace PM1.1 and PM1.2 about the access conflict, and offer them three options to solve the problem:

1. **Only one actor is allowed to update the process fragment:** Initiate an negotiation about who is going to get the access to update the process fragment **Parallel-Experiment**. If one of the involved parties does not want this kind of negotiation, one of the solutions below must be used.
2. **Synchronise the changes:** Initiate *coordination agents*, that will help the users synchronise their changes by making changes visible for all involved actors. If some changes are conflicting, negotiation agents are used to solve these situation. The synchronisation is supported through merge tools that identify conflicting parts of the XML-file defining the process fragment.
3. **Synchronise when finished:** This means that both workspaces updates the process fragment independently. However, when when the process fragment is checked back to the repository through the workspace manager, changes most be merged or a new separate version of the fragment must be created.

Another situation we have to take care of, is when changes are made to e.g., two inter-related process fragments in two different workspaces. This situation can be dealt using the same three solutions as shown above. One of the involved workspaces can give away update access to a process fragment, the changes in the two workspaces can be synchronised (meaning the the changes made to a process fragment in one workspace, will be visible in the other and vice versa.

19.4.2 Support for scenario 2

In scenario 2, only one workspace could change a specific process fragment at a time. Here we have to deal with, how the changes of a process fragment should be propagated to the rest of organisation. When a process fragment has been updated, all other workspaces that have checked out the same process fragment for read, must be notified. The workspace manager will activate *update agents* notifying all involved workspaces that the process fragment has been updated, and ask them if they want to use the new version of the process fragment. If the organisation has as a policy to always use the most recent version of process fragments, the users in workspaces will be forced to update the process fragment by the update agents. There are basically three choices of update policy:

1. **Ignore update** Involved workspaces will be notified, but the process fragments will not be updated. This means that the changes of process fragments will only be propagated when a new similar project is started.
2. **Update at will** It is up to the users if they want to update the process fragment or not. This means that different versions of the process fragment is allowed, and could cause some consistency problems.
3. **Forced to update** It will always be the most recent version of process fragments used.

19.4.3 Support for scenario 3

Here we have to solve the problem of propagating updated versions of inter-related process fragments to workspaces having older versions of the same process fragments. The solution is basically the same as described in section 19.4.2, but either all of the inter-related process fragments are updated or none are to ensure consistency. If this policy is not followed and only a few of the involved process fragments are updated, negotiation agents will be used to decide if the update should be cancelled or if every involved process fragment should be updated.

19.4.4 Support for scenario 4

When there is only one person that have access (both read and write) to a process fragment at time, to consistency problems can occur. The biggest problem with this approach is that process fragments will be totally locked for a time. When a user in a workspace wants to check out a particular process fragment that is exclusively checked out by another, the workspace manager will send a *notification agent*. The notification agent will give information about who has checked out this process fragment, when it was checked out. The notification agent could also be triggered by the user, to check for how long this process fragment will be locked. The notification agent will then ask the user that has access to the process fragment (or an agent representing this user) about when (s)he believed to be finished by the job.

19.4.5 Support for scenario 5

In this scenario, we looked at what happened if different access policies were used in an organisation. A central result in scenario was to keep consistency by forcing a workspace to have a access policy not violating its parent workspace according to the four levels of strictness as listed in section 19.3.5.

If at one level in the workspace hierarchy somebody decides to go for a more strict access policy, all the sub-workspaces, must adhere to this change. If this organisation is based on democratic principles, the workspace manager when being alerted about the change of access policy, will activate *voting agents*, that will ask all involved workspace to change to a more strict access policy or not. Based on the result of this voting, the access policy will be changed or not. If democratic principles are not used, people can be forced to change access policy. In any case if a change will take place, all the involved parties will be notified by *notification agents* and be asked if they are ready to change access policy. This is done to ensure that the all involved parties has adopted the new access policy (checked in process fragments etc.), and is ready to change.

If at one level in the workspace hierarchy, somebody wants to go for a less strict access policy, this will not affect the sub-workspaces but rather workspaces on a higher level. A negotiation process must be initiated in order to get an agreement to go for a less strict access policy. In a democratic organisation, *voting agents* can be used to reach an agreement. For a more strictly hierarchical organisation, the manager at the top could make the decision to make a change of access policy or not. If the access policy is changed, affected workspace must be notified by *notification agents*, but there is no need to wait for people to adapt to the new policy. People can work as before, or they can update process fragments in a less strict manner.

19.5 Implementation of awareness agents

So far we have only identified the types of agents that could be used to provide awareness support for workflow consistency. These agents are negotiation agent, coordination agent, update agent, notification agent, and voting agent. These agents will communicate in Agent Meeting Places (AMPs) using KQML to specify the communication. We have defined the following performatives (speech-acts) that are used by agents: *ask-if, tell, untell, register,* and *unregister*. The register and unregister performatives are used to register to AMPs. Our multi-agent architecture, offers a high-level Java API used to program agents. The API provides methods for moving agents between AMPs and workspaces, communicates with other agents, (un)register in AMPs, clone agents etc. Graphical user interfaces for agents are supported through Java Swing classes, and our multi-agent architecture offer graphical user interfaces to manage agents in workspaces, and administration of AMPs.

19.6 Conclusion

In this paper we have looked at how software agents can be used to provide awareness support for solving consistency problems of distributed workflow models. We have identified the necessary agents, and the functionality these agents should provide. Today, we have only programmed simple prototypes of these agents, and they are not fully integrated into the whole CAGIS PCE. The workspace manager has also to be completed, before our approach can be more extensible tested. These tests will be based on real-life industrial scenarios, and involve modelling of workflow, as well as implementation of awareness agents. Future research will give more detailed experiences using software agents for providing workflow model consistency.

Acknowledgement

I would like to thank Reidar Conradi for giving useful comments on the paper.

CHAPTER 20

Evaluation of a Cooperative Process Support Environment

Alf Inge Wang[1]

Abstract
This report describes an evaluation where the same distributed conference organising process is modelled in three different process centred environment Endeavors, ProcessWeb, and our own CAGIS Process Centred Environment. Endeavors is an activity based, flexible workflow system, ProcessWeb is a role-based workflow system with a web-interface, whereas the CAGIS Process Centred Environment combines an activity based workflow system with a software agent system. The goal of the experiment is to investigate if a combination of a traditional workflow system and software agent system better can model and support distributed cooperative processes than stand-alone workflow systems. We also want to investigate if a combination of workflow system and agent system better can adapt to occurring process changes. A conference organising scenario is used as a case in the experiment because it illustrates a distributed process containing both simple, individual activities as well as more dynamic, cooperative activities. By evaluating how well the different process centred environments can model and support the scenario, as well deal with process changes, our CAGIS Process Centred Environment is validated.

[1]Dept. of Computer and Information Science, Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway. Phone: +47 73 594485, Fax: +47 73 594466, Email: alfw@idi.ntnu.no

Keywords: Process centred environments, Process modelling, Evaluation, Cooperative activities.

20.1 Introduction

In 1997, the Norwegian research project called Cooperating Agents in the Global Information Space (CAGIS) was initiated. One of the goals for the CAGIS project was to create models and architectures to support distributed, cooperative processes. The prototype CAGIS Process Centred Environment (PCE) is a result from this research. The CAGIS PCE consists of three main parts, an activity-based workflow system, a multi-agent system, and a gluing-framework for specifying and executing the interaction between the workflow system and the multi-agent system. Activity-based workflow systems are very good at modelling and supporting repetitive, structured processes containing a network of activities with simple data- or control-flow relations. On the other hand, software agents are ideal to model cooperative activities [JLT99, BvET97, TGML98, CMM97, MM97]. By combining activity-based workflow with software agents in CAGIS, we believe that processes containing cooperative activities can be better supported. In this report, we distinguish between two types of activities: Individual and cooperative activities. Individual activities are activities where one role is assigned for performing this activity. In cooperative activities, several roles are involved in performing the activity.

This report compares three PCEs according to their ability to model and support a conference organising process, and how process changes to this particular process can be dealt with. With this evaluation we want to validate if our approach of combining traditional workflow with software agents can do more than a stand-alone workflow system. The other workflow system chosen for this experiment are Endeavors and ProcessWeb. Endeavors is a flexible activity-based workflow system capable of dealing with dynamic process changes. ProcessWeb is a role-based workflow system offering a web-interface to the user. ProcessWeb's domain is modelling of interactions between actors in a process. These two workflow systems is chosen because of their flexibility and that they represent a different process modelling philosophy.

The text *Experimentation in Software Engineering* by Wohlin et. al [WRH$^+$00] has been used as a starting point to formulate the evaluation. The method used is based on the Goal Question Metric (GQM) method [BCR94b]. The case chosen is a conference organising process. Originally we wanted to use a cooperative software engineering case from the software industry, but due to not being able to find one described in sufficient detail, the conference organising process was selected. The advantage selecting this case, is that the process is widely known and described ([OSVS82, Car97]) and contain both individual, structured activities as well as more dynamic, cooperative activities.

20.2 Definition of Experiment

In this section we will define experiment used to evaluate our CAGIS PCE. The term experiment will in this report denote an ad-hoc experiment. The validation method used in this report is, according to Zelkowitz and Wallace in [ZW98], the assertion validation method where developers being both experimenters and subjects of study.

20.2.1 Goal definition

The *goal definition* is the first step to analyse the problem at hand. In this case, the objective of the experiment is to compare three different process technologies to see how well these technologies can support processes that both have individual as well as cooperative activities. Also it is important to investigate how well the three different process technologies adapt to process changes.

The motivation for the experiment is to see if a new approach used in our CAGIS PCE, is a feasible and perhaps better way to model and support processes containing both individual as well as cooperative activities. The new approach used in CAGIS PCE, combines traditional workflow with software agents.

Object of study

The object of this study is the three process technologies Process Web, Endeavors and CAGIS PCE, and their ability to model and support a distributed conference organising process.

Purpose

The purpose of the experiment is to compare the ease of modelling and performance of the three process technologies in modelling and supporting a distributed conference organising process. The distributed conference organising process consists of both structured, individual activities as well as dynamic cooperative activities, and the performance will be measured to see the coverage of these two aspects. In addition we will investigate how the different technologies can cope with process changes.

Perspective

The perspective is from the point of view of the researchers, and the process is defined in the researchers organising conferences [OSVS82].

Quality focus

The main effect studied in the experiment is model coverage of structured, individual activities and cooperative activities, and support for dynamic aspects of a process.

Context

The evaluation was run by Alf Inge Wang at the Department of Computer and Information Science, at the Norwegian University of Science and Technology in Trondheim, Norway. Mark Greenwood, in Informatics Process Group at the University of Manchester has also done some modelling of cooperative activities in ProcessWeb, and Professor Richard N. Taylor, Department of Information and Computer Science at the University of California, Irvine has suggested a solution to model cooperative activities for Endeavors. Most of the modelling (apart from Greenwood's effort ProcessWeb) and the measurement is done by the author.

20.2.2 Summary of definition

Comparing *three different process technologies*
for the purpose of *evaluating models*
with respect to *support for structured, individual activities and cooperative activities, and to handle dynamic changes of such processes*
from the point of view of the *researchers*
in the context of *research community*.

20.3 Planning of the Experiment

This section describes planning necessary to execute the experiment.

20.3.1 Context selection

The execution of the experiment has run at the University and will therefore not be evaluated in an industrial setting. This means that the result of the experiment is not automatically valid in an industrial setting, but the result should give an indication of what to expect. The experiment addresses a real problem, i.e., how to give sufficient process support to both structured, individual activities as well as cooperative dynamic activities.

The description of the experiment should provide enough information for other researchers to replicate the experiment in the same or the actual environment.

20.3.2 Research Questions

There are two research questions we would like to address in this experiment: Is a combination of a traditional workflow system and a software agent system better compared to a stand-alone workflow system to:

1. Model and demonstrate enactment of processes containing both dynamic cooperative activities, as well as structured, individual activities ?

2. Adapt to process changes ?

20.3.3 Evaluation of the research questions

The evaluation of the two research questions listed above will be answered in a combination of a quantitative research and a qualitative research method. By quantitative means, research question one will be measured by looking at how much of (coverage) the conference organising process is modelled and enacted by the process model in each PCE. The *coverage* will be measured in number of activities that can be modelled and enacted. The scale 1-5 will be used to express how much of an activity is modelled. If every aspect of an activity can be modelled it will be weighted by 5. If not any aspect of an activity can be modelled, it will be weighted by 0. The rest of the scale 1-4 will be used to say how much of the activity can be modelled. The conference process consists of 25 activities, and the percentage coverage of the model will measured by: (score /25 x 5) * 100. In addition will the developing time spent on creating the process models in the three process environments be measured.

Further research question two will be measured by measuring *adaptability*. Adaptability is measured, by assessing the effort spent on implementing a specific process change. A scale 1-5 will be used to indicate how much effort has been used. 5 indicates *Very little effort*, 4 indicates *Little effort*, 3 indicates *Some effort*, 2 indicates *Strong effort*, and 1 indicates *Very strong effort*.

As the statistical data for this experiment is insufficient because only one scenario is modelled, qualitative discussions will also be used to to evaluate the research questions when comparing the results from the experiment. The empirical data will be used to give an indication of the differences, and reasons for these differences will be given when evaluating the experiment.

20.4 Three Process Centred Environments

This section describes the three process centred environments (PCEs) we used in the evaluation. The first PCE, Endeavors, is a semi-commercial workflow tool usually used to model business processes. The second PCE, ProcessWeb, is a web-based research prototype based on a commercial process engine used to model business processes as well as software development processes. The last PCE presented, is our own CAGIS research prototype that combines software agents with workflow, and can be used to model business processes as well as software development processes.

20.4.1 Endeavors

Endeavors is an open, distributed, extensible process execution environment developed at University of California Irvine, and has been licensed by Endeavors Technology Incorporated. It is designed to improve coordination and managerial control of development teams by allowing flexible definition, modelling, and execution of typical workflow applications. There are five main characteristics for Endeavors:

- **Distribution** Support for transparently distributed people, artifacts, process objects, and execution behaviour (handlers) using web protocols.
- **Integration** Allows bi-directional communication between its internal objects and external tools, objects, and services through its open interfaces across all levels of the architecture. Multiple programming languages are also supported through APIs.
- **Incremental Adoption** Components of the system (user interfaces, interpreters, and editing tools), may be down-loaded as needed, and no explicit system installation is required to view and execute a workflow-style process.
- **Customisation and Reuse** Implemented as a layered virtual machines architecture, and allows object-oriented extensions of the architecture, interfaces, and data formats at each layer.
- **Dynamic Change** Allows dynamic changing of object fields and methods, the ability to dynamically change the object behaviours at runtime, and late-binding of resources needed to execute a workflow process. Process interpreters are dynamically created as needed.

The Endeavors architecture is object-oriented and consists of three major levels: The user level, the system level, and the foundation level. The *user level* is responsible for maintaining consistent views to users through management of coordinated updates. The *system level* maintains the category object model abstractions and data structures, while the *foundation level* is responsible for storing objects and invocation of handlers. In figure 20.1 the architecture of Endeavors user level and partly the system level is shown. The user interact through a *Client Manager* that can invoke different artists (views) through the *Artist Manager*. Instances of different artists communicate with a System Level Interface wrapper and a Client Event Dispatcher (at the system level). More details about the Endeavors architecture can be found in [HL98].

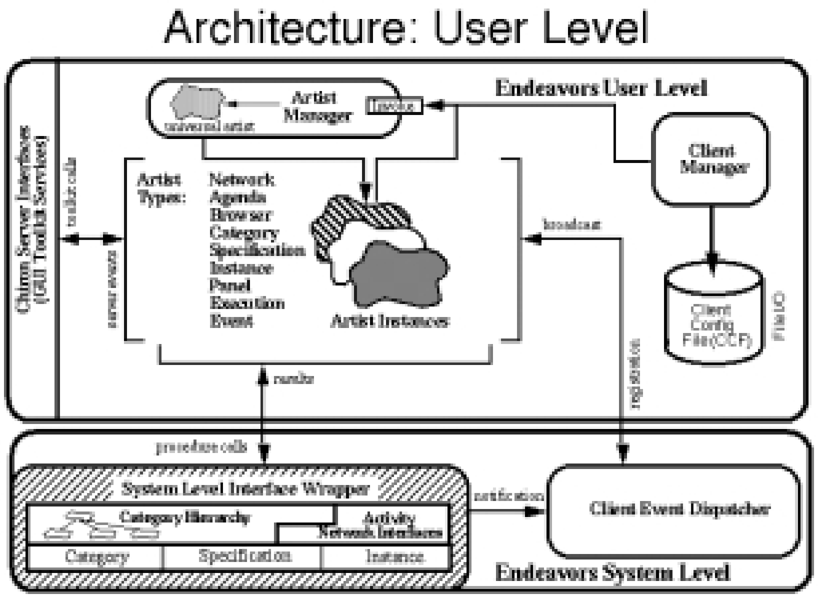


Figure 20.1: Endeavors architecture at user-level

Endeavors Process Modelling Language

The Process Modelling Language (PML) used in Endeavors is object-oriented, based on the Teamware process modelling language [You94], and consists of five major modelling *categories* [Gre98]:

- **Activity** An Endeavors activity is an end-user or automated agent-specific task that can have the state *initialized, enabled, executing, completed, and terminated*. An activity specification will take in a list of input artifacts, use the resources it has to process, create new or modify old artifacts, and then pass off the newly created/modified artifacts to its output list. Activities may also have sub-activities or networks of sub-activities.
- **Artifact** An artifact is anything that can be produced or consumed by an activity, typically a document. The top level Endeavors artifact defines the methods for share, open, lock and restore artifacts.
- **Resource** Resources represents standard project management resources such as personnel, meeting rooms, budgets, computers. The most commonly used resource in Endeavors are software tools such as compilers, and Web browsers.
- **Network** Networks are used in Endeavors to provide abstraction by logically grouping inter-dependent activities. Start and finish activities represent the first and final actions of a process. Fork and join activities are used to specify activities executed

in parallel. Branch and merge activities are used to specify conditional activity flow (can for instance be used to implement feedback loops).

- **Arc** An arc is used to define relationships between the categories listed above. Arcs can represent control, data, or resource flow. Arcs have no default behaviour associated with them, but assist in determining the well-formedness of a workflow fragment.

Each of these categories has their own set of messages, definition of fields, and default handlers, but each definition can be dynamically extended, altered, assigned, removed, or re-specified.

Processes can be modelled in Endeavors using the graphical Network Artist tool enabling the user to draw processes using a palette of activities and control-flow mechanisms. If the standard available activities are not sufficient, user-defined activity types can be created. When an activity is activated, a handler is executed. A handler is a program using Endeavors API to communicate with the system. Handlers can be used to wrap commercial tools, or the user can create his/her own tools (questionares, simple user-interfaces etc.). Some system handlers are available, and the user can create his/her own handlers using one of the supported programming languages Java, Python, Ada95, or Tcl.

20.4.2 ProcessWeb

ProcessWeb [Yeo96] is a web-interface based workflow system based on the ProcessWise [PMC96] Integrator (produced by ICL) implemented by Information Process Group, University of Manchester. The web-interface is provided through the ProcessWise Integrator application interface. ProcessWise Integrator creates an environment enabling the activities of people in an enterprise to be coordinated and integrated with the organisation's computing facilities. A process management system built using the ProcessWise Integrator has a client/server structure and consist of four main components: User Interface, Process Control Manager, Process description in PML, and an Application Interface. Figure 20.2 shows an illustration of how these components are related.

The most important component of ProcessWise Integrator is the Process Control Manager (process engine), which acts as the central server. Its main function is to interpret the PML description of the process. To ensure that processes may continue indefinitely, the Process Control Manager has been implemented using a persistent store technology.

ProcessWeb Process Modelling Language

ProcessWise Integrator's PML is also object-oriented, and uses roles as the main concept for modelling processes. Objects are called *roles*, and *interactions* to provide communication channels between roles. The role concept helps address the complexity of models by providing a quasi-intuitive structuring of activities (human processes are messy and might not be amenable to hierarchical approach). A role is defined by its *actions* (methods) and

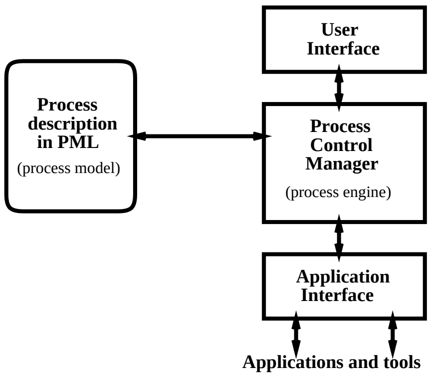


Figure 20.2: Structure of ProcessWise Integrator

its *resources* (attributes). Preconditions called *'when' guards*, are used to select action for enaction, and they are expressed as if statements. Interactions are uni-directional, asynchronous typed communication channels provided through a takeport and a give port. A *takeport* received data or control flow from another role, and a *giveport* sends data or control flow to another role. The giveport and takeport represents two ends of a single interaction. No global variables can be defined in PML.

Figure 20.3 shows a simple example of a role called *John*. If this role receives the message "Hello John!", the message "Hello to you too!!!" will be sent back to the sender. Note that PML to specify the connection to the other role is note presented here. The role is defined by a set of *resources*, defining the variables used in the role, and a set of actions (only one action for this role). The PML consists of ordinary programming expressions like *if-then*, and there are as well some predefined methods like *Take* and *GiveCopy*. The syntax for a method is *Methodname (parameter1 = variable1, parameter2 = variable2...)*. Usually actions are specified with pre- and post-conditions (not shown in figure 20.3).

20.4.3 CAGIS Process Centred Environment

This section describes the CAGIS Process Centred Environment (PCE). The CAGIS PCE consists of three subsystems:

- **CAGIS SimpleProcess** is a workflow tool with a web-interface, used to model stable processes consisting of networks of activities that are executed by individuals. For users of CAGIS SimpleProcess, the activities will be shown as web-pages that can contain a work description, links to relevant document or tools, HTML-forms for entering necessary information, or a Java applet. A process in CAGIS SimpleProcess can consist of several autonomous process fragments that can be distributed over several workspaces. A process fragment can be anything from one single activity to several related activities. The CAGIS SimpleProcess workflow

```
John isa Role with
resources
        gp: giveport String
        tp: takeport String
        message: String
        sendmessage: String
        i: Int
actions
  init: {
         Take (interaction = tp, gram = message);
         if message := "Hello John!"
         then
                 sendmessage ="Hello to you too!!!";
                 GiveCopy (interaction = gp, gram=sendmessage);
         end if
       }
end with
```

Figure 20.3: The role John described in PML

tool offers an *agenda-tool* for web users, to make it easier to know the state of the process, what tasks to do next etc. The process models can be written as textual XML-documents, or the web-based *process modeller-tool* can be used to enter activities directly. In the former approach, the XML-document defining the process model must later be parsed, and the process will then be instantiated. In the latter approach, the activities will be instantiated as the user enters activity information into the tool (activities will be instantiated incrementally one-by-one). A *state server* (the process engine) will take care of the process states, and a *monitor tool* is used by managers to supervise the progress and state of the process. Hierarchical workspaces (i.e., a tree structure) are used model the organisation executing the process, and CAGIS SimpleProcess has support for moving process fragments between workspaces (between users) and also between sites. A more detailed description of this workflow tool is found in [Wan00b].

- **CAGIS Distributed Intelligent Agent System (DIAS)** takes care of cooperative activities between workspaces in the CAGIS PCE framework [WLC99, PHBN99, HN00]. The software agents in CAGIS DIAS can be used to coordinate artifacts and resources, negotiate about artifacts and resources, monitor the working environment for changes or events, provide infrastructure for brainstorming, electronic meetings, trading services etc. CAGIS DIAS provides the infrastructure for creating cooperating agents, and consists of four main components:

  - **Agents:** are autonomous software acting on behalf of a user or some other agent, set up to achieve a modest goal. We have defined three types of agents: *System agents* - give support to other agents, *Interacting agents* - provide means for cooperation between users and agents, and *User agents* - agents set to help users with specific problems.

  - **Agent Meeting Places (AMPs):** are where agents meet and interact, controlling and providing facilities for how the agents can exchange information and services.
  - **Workspaces:** are temporary containers of relevant data where agents interact with users and tools. Workspaces uses repositories as persistent storage of files.
  - **Repositories:** are persistent storages where any information can be stored or retrieved. Repositories can be accessed directly by agents, and special repositories can be used as Experience Bases for software agents.

- **CAGIS GlueServer** To enable CAGIS SimpleProcess to interact with the CAGIS DIAS, we have built a CAGIS GlueServer. The CAGIS GlueServer is a middleware that uses a so called GlueModel, where relations between process fragments and software agents are defined. The GlueModel can be seen as a part of the process model defining rules for interaction with others, but also as a meta-model since it can specify changes of the process model. The GlueModel can specify three types interactions between a process fragment and an agent:

  - **Predefined Interface:** An agent is executed instead of a process fragment, e.g., a negotiation activity delegates its job to a negotiation agent.
  - **Periodic Invocation:** Upon termination of a process fragment, an agent is used to help decide what to do next. Possible decisions include: Go to next process fragment, iterate current process fragment, change and re-execute current process fragment, or move current process fragment to another workspace.
  - **Dynamic Monitoring:** A monitoring agent probes events and status in the working environment. When a specified event occur, a relevant process fragment may be executed.

  By using a *separate model* to describe the interconnection between agents and process fragments, it is possible to use other workflow tools (e.g. than CAGIS SimpleProcess) as well as other agent systems. This makes the CAGIS PCE more open-ended, and ready to meet future challenges. The GlueServer and the GlueModel are described in further detail in [WCL00].

CAGIS Process Modelling Languages

To specify a process in CAGIS PCE, two different process models and an agent-API are used, according to the sub-systems described above. Here is a short description of the two model and the agent-API:

CAGIS SimpleProcess PML

The CAGIS SimpleProcess PML is used to model simple, individual, repeatable processes. The main building block is activity. An activity is represented by a web-page that

```
<Activity>
  <Name>Edit resource-report</Name>
  <Workspace>Prod Dept/Project Manager</Workspace>
  <Prelink>Prod Dept/Project Manager/Read resource-report</Prelink>
  <Postlink>Prod Dept/Project Manager/Send resource-report</Postlink>
  <Description>Make changes to the resource-report
     based on personal opinion</Description>
  <Code>
    <HTML>
      Remember this:
      <li>Do not change the first section</li>
      <li>Make a new version of the report with your changes</li>
    </HTML>
  </Code>
</Activity>
```

Figure 20.4: An example of an activity modelled in CAGIS SimpleProcess PML

has a state and relationships to other web-pages. Since activity models are quite similar to web-pages, the terms *Prelink* and *Postlink* are used to identify ordering relationships between activities. If the activity C has the activities A and B as *Prelinks* and the activity D as a *Postlink*, the sequence of activities will be $(A, B) \Rightarrow C \Rightarrow D$, where A and B may be executed in parallel. Activities are placed in hierarchical workspaces used to model the organisation. Workspace names can be used to identify a company, department, project group, roles, or even specific users. Workspaces can be private or shared. For each workspace, an *agenda* will show activities related to this workspace. Several instances of the same activity can be initiated in different workspaces. If several instances are needed in the same workspace, different activity name is used for each instance. The *code* (script) part of the CAGIS SimpleProcess PML is simply HTML, that can be used for describing information, get input from the user through a HTML-form, or to execute a Java applet.

The CAGIS SimpleProcess PML is specified in XML, and an example with one activity is shown in figure 20.4.3.

From the example, we can see that an activity is identified by a *name*, a *workspace*, and a *description*. In addition *prelinks* and *postlinks* are used to describe relationships to other activities, and a *code* part defines the presentation or action of the activity. The code can also simply be a URL to an HTML-file, or be used to invoke an Java applet.

DIAS Agent Language

To support dynamic processes, the DIAS Agent Language is used. DIAS Agent Language is a high-level agent API written for Java. KQML is used to define how agents should communicate, and the content of a speech-act (e.g. tell-agent, ask-agent etc.) is specified in XML. It is possible to implement both stationary as well as mobile agents. Examples of agent methods available are: migrate agent, communicate to other agent, tell other agent, announce service, request service, etc.

Glue Modelling Language

The Glue Modelling Language is used to specify how process fragments specified in SimpleProcess PML and software agents (in DIAS) interact. The language has again an XML syntax, and specifies the relationship between a process fragment and a agent.

```
<FRAGMENT-AGENT-PAIR>
 <Agent agent-class="agent.Negotiation" amp-id="CompanyAMP">
   <Interaction-Type>negotiation</Interaction-Type>
   <Result>Yes|No</Result>
 </Agent>
 <Fragment fragment-id="MPG/estimate-resources">
  <Reaction>
     <Result>Yes</Result>
     <Action fragment-id="MPG/allocate-resources"
             body="execute_process_fragment_PFNUMBER">
     </Action>
     <Result>No</Result>
     <Action fragment-id="MGP/estimate-resources"
             body="change_and_reexecute_process_fragment_PFNUMBER">
     </Action>
  </Reaction>
 </Fragment>
</FRAGMENT-AGENT-PAIR>
```

Figure 20.5: An example of a process fragment - agent pair specified in a GlueModel

In figure 20.4.3, a small example of a GlueModel containing one process fragment - agent pair is shown. The GlueModel consists of two main parts: An *Agent* part and a *Fragment* part. The Agent part is described by an agent class, an identification for where the agent will interact with other agents (amp-id), the type of interaction involved, and the result expected from the agent. The Fragment part is specified by a process fragment identification (workspace name and process fragment name), and a *Reaction*. The reaction consist of two main parts: a *Result* and an *Action*. The Result is similar to an ”if-expression” in a programming language and specifies an expected result returned from the agent. The Action part specifies what to be executed in the workflow tool, the GlueServer or in the agent system.

20.4.4 Process Centred Environment Summary

In this section, three different process environment have been presented by describing the overall architecture, tools offered, and the process modelling language used. We would now like to compare these environments.

In table 20.1 describes a comparison of the three PCEs. These notes should be made about the table. In *feature* 3, Endeavors expresses the PML graphically, but Java handlers must also be written to get the activities do something useful. In *feature* 5, the table shows whether the PCEs supports user-interfaces packages provided in programming languages (e.g., in Java). In ProcessWeb and in CAGIS SimpleProcess, Java user-interfaces can be provided by invoking Java applets in the HTML-code. The agent system in CAGIS PCE can use Java user-interfaces directly.

Feature	**Endeavors**	**ProcessWeb**	**CAGIS PCE**
1. Activity based PML	Yes	No	Yes
2. Role based PML	No	Yes	Yes (agents)
3. PML expressed as	Graphical model	Text	Text
4. Web User-interface	Indirectly	Directly	Directly
5. Native user-interface	Yes	Indirectly	Indirectly/directly

Table 20.1: Summary of features in the three process centred environments

The main difference between the three PCEs is the way the processes are modelled and the philosophy for the process modelling language. In Endeavors and in CAGIS SimpleProcess, the process modelling language specifies the process mainly through activity networks. In ProcessWeb the process models are specified by modelling roles that interacts. The CAGIS PCE also allows the modelling of role interaction through the agent system (CAGIS DIAS). We can therefore we say that the CAGIS PCE uses a mixed PML philosophy.

The way the processes are modelled differs also between the three environments. In Endeavors the process is modelled through manipulation of graphical objects in addition to writing activity handlers in some programming language. In ProcessWeb, any text-editor can be used to write PML. In the CAGIS SimpleProcess, the PML can be written in a text-editor as an XML-document, or a simple HTML-form based tool. The agents are implemented in any Java programming environment.

ProcessWeb and CAGIS SimpleProcess offers user-interfaces directly through the Web-browser. In addition, Java applets can be invoked indirectly in the HTML-code. In Endeavors, the user interface is dependent on the handlers that can be implemented in Java, Python, Ada or Tcl. This means that the user-interfaces typically can be implemented using standard GUI packages provided in the programming languages. A Web-interface could also be provided indirectly, e.g. through some Java code. The agents in CAGIS DIAS uses Java GUI-packages to provide a user-interface.

20.5 Actual case example: The conference management process

Here we describe in detail a conference organising process, to be used as a case for evaluation. The scenario was chosen because it contained some cooperative elements as well as simple coordination, and it is a well-documented scenario. In Olle et al. [OSVS82], a short textual description of the conference organising process is presented. A more detailed description of the same case can be found in [Car97]. We describe the scenario using a simple graphical notation showing activities and their sequence, and an additional textual description of each activity (also decompositions of activities).

The high-level process of the conference organising process consists of seven activities as modelled in figure 20.6.

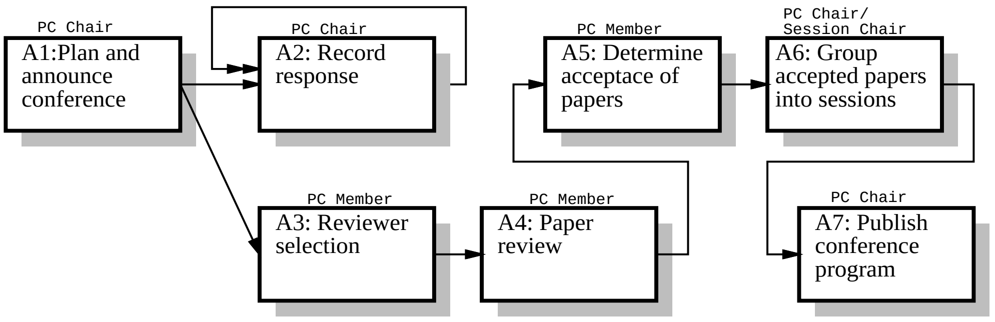

PC Chair
A1:Plan and announce conference
PC Chair
A2: Record response
PC Member
A5: Determine acceptace of papers
PC Chair/ Session Chair
A6: Group accepted papers into sessions
PC Member
A3: Reviewer selection
PC Member
A4: Paper review
PC Chair
A7: Publish conference program

Figure 20.6: A High-level model of the conference organising process

The rest of this section gives a more detailed description of each of these activities. A decomposition of the conference process is shown in figure 20.7. The notation used has been developed by the author, and is an extension of a graphical notation used to show process models in the EPOS PSEE. The extensions added to the original modelling language is ability to explicitly show decomposition of activities, show cooperative activities, and graphical notation for control flow. Section 20.5.10 describes all documents and databases that are accessed in the conference scenario.

20.5.1 A1: Plan and announce conference

```
Function:        Issue call for papers to potential participants.
Responsible:     PC Chair
PreCondition:    None
PostCondition:   A2 (Terminated by Program Chair)
Document access: Conference-personDB(R/W), Call-for-papers(R/W)
```

R and W in the above table show how activities access documents: R refers to read, while W refers to write).

A1 has the following three sub-activities (A1.1 and A1.2 are done in parallel):

- **A1.1: Make call for papers** Based on an available *Call-for-papers* template, a new call for papers will be created. The activity should access an editor where Call-for-papers can be written.

- **A1.2: Manage person information** In this activity, the PC Chair must search for persons (the activity should provide access to the DBLP bibliography servers[2] and the ResearchIndex[3]) to update the database for potential participants. An interface should also be presented to enter data in the Conference-personDB
- **A1.3: Distribute call for papers** Based on the results from A1.1 and A1.2, Call-for-papers will be distributed to potential participants (via email). PreCondition: Finished A1.1 and A1.2.

20.5.2 A2: Record response

```
Function:         Handle response from potential participants.
Responsible:      PC Chair
PreCondition:     Finished A1
PostCondition:    None
Document access: Email/mail(R), Conference-personDB(R/W), Acknowledge-templates(R)
                  Conference-papers(R), Paper-list (R/W)
```

A2 has the following sub-activities (based of the response received in A2.1, one of the three activities A2.2, A2.3 or A2.4 will be next):

- **A2.1: Check response and Maintain personDB** First, this activity has to check email sent to the conference email address (e.g., response@conference.org). Then the *Conference-personDB* will be updated to track responses.
- **A2.2: Handle Received paper** Register received paper in *Conference-personDB* and *Paper-list* (including title, address, authors, keywords etc), and store paper electronically. Then use a template to send an acknowledgement to the author.
- **A2.3: Acknowledge Letter of Intent** Send an acknowledgement and use an available template for this.
- **A2.4: Regret late response** Use a standard available template to send a regret for a late response.

20.5.3 A3: Reviewer selection

```
Function:         To select specific reviewers for each particular paper
Responsible:      PC Member
PreCondition:     Finished A1
PostCondition:    A4
Document access: Paper-list(R/W), Conference-personDB(R), Conference-paper(R).
```

In A3, the Conference-personDB will be used to find the PC Member's ID. A3 further has two sub-activities:

[2]http://www.informatik.uni-trier.de/∼ley/
[3]http://citeseer.nj.nec.com/cs

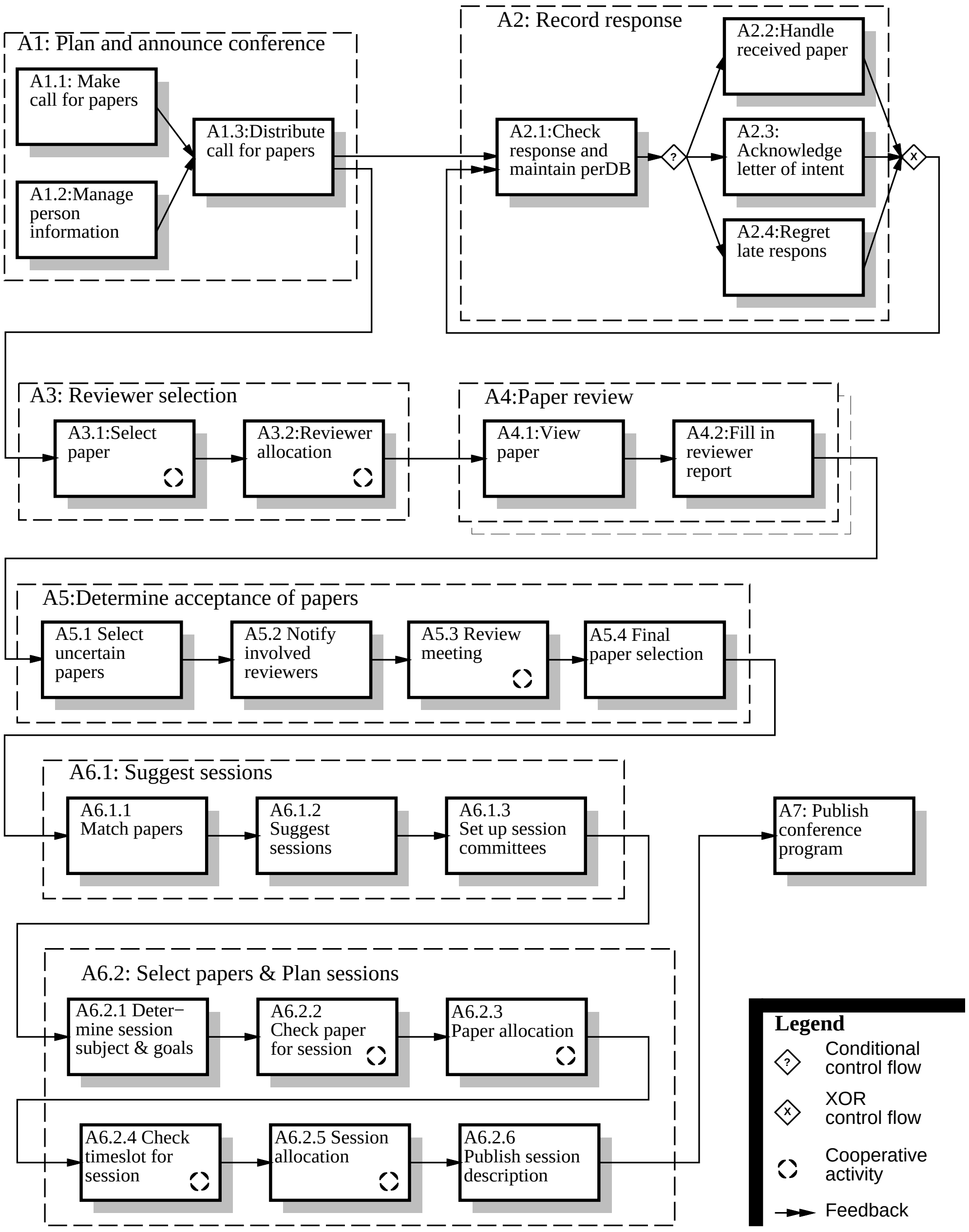

A1: Plan and announce conference
A1.1: Make call for papers
A1.2:Manage person information
A1.3:Distribute call for papers
A2: Record response
A2.1:Check response and maintain perDB
A2.2:Handle received paper
A2.3: Acknowledge letter of intent
A2.4:Regret late respons
A3: Reviewer selection
A3.1:Select paper
A3.2:Reviewer allocation
A4:Paper review
A4.1:View paper
A4.2:Fill in reviewer report
A5:Determine acceptance of papers
A5.1 Select uncertain papers
A5.2 Notify involved reviewers
A5.3 Review meeting
A5.4 Final paper selection
A6.1: Suggest sessions
A6.1.1 Match papers
A6.1.2 Suggest sessions
A6.1.3 Set up session committees
A7: Publish conference program
A6.2: Select papers & Plan sessions
A6.2.1 Determine session subject & goals
A6.2.2 Check paper for session
A6.2.3 Paper allocation
A6.2.4 Check timeslot for session
A6.2.5 Session allocation
A6.2.6 Publish session description
Legend
Conditional control flow
XOR control flow
Cooperative activity
Feedback

Figure 20.7: Decomposition of the conference process.

- **A3.1: Select paper** All PC Members will receive a list of submitted paper with hyper-links that makes it possible to read through the papers. The PC Member will then mark papers they are interested in reviewing. The PC Members' ID will be added to the Possible-reviewer-list in Paper-list.
- **A3.2: Reviewer allocation** All papers should be reviewed by three reviewers. If more than three reviewers are allocated, a negotiation process between the involved reviewers is initiated. If some papers do not have enough reviewers (less than three), PC members with fewest marked papers will be asked to review these papers. See section 20.5.8 for details for how to handle the negotiation process. When all papers have three reviewers, the Actual-reviewer-list in Paper-list will be updated to reflect this change. Activity A3.2 will not finish until three reviewers are allocated to all papers.

20.5.4 A4: Paper review

```
Function:        Review the papers according to the specifically prepared review
                 reports.
Responsible:     PC Member
PreCondition:    Finished A3.2
PostCondition:   A5
Document access: Conference-personDB(R), Paper-list(R), Conference-papers(R),
                 Review-reports(R/W)
```

A4 has two sub-activities (A4.1 and A4.2 are executed sequentially for every paper reviewed):

- **A4.1: View paper** View the reviewed paper along with reviewing instructions. The papers to be reviewed can be found by finding PC Member ID in Conference-personDB and search for this ID in the Paper-list.
- **A4.2: Fill in reviewer report** Fill in a web-based form to generate a Review-report. Each Review-report will be identified with the reviewer's ID and a paper ID.

20.5.5 A5: Determine acceptance of papers

```
Function:        To perform final accepted papers selection after a finished review
                 process.
Responsible:     PC Member / PC Chair
PreCondition:    Finished A4
PostCondition:   A6
Document access: Paper-list(R/W), Conference-personDB(R/W), Conference-papers(R),
                 Acknowledge-templates(R), Review-reports(R)
```

Sub-activities (sequence A5.1, A5.2, A5.3, and A5.4):

- **A5.1 Select uncertain papers** Responsible: PC Chair. Look through review results and update *State* and the *Review-result* in Paper-list. The state can be Rejected, Uncertain, or Accepted. The Review-result will reflect the average review score for the paper.
- **A5.2 Notify involved reviewers** Responsible: PC Chair. Notify involved reviewers to look through the papers with the state *Uncertain* one more time and prepare review-meeting.
- **A5.3 Review meeting** Responsible: PC Members. The review meeting can be held using Internet Relay Chat (IRC), net-meeting or similar tools.
- **A5.4 Final paper selection** Responsible: PC Chair. Update the *Paper-list* and *Conference-personDB* to reflect the results from A5.3 above (all papers that had the state *Uncertain*, will now either have the state *Accepted* or *Rejected*). Acknowledgement of paper accept/reject will be sent to all authors (standard conference template will be used for this purpose).

20.5.6 A6: Group accepted papers into sessions

```
Function:         To finalise the conference program through grouping accepted
                  papers into session.
Responsible:      PC Chair / Session Chair
PreCondition:     Finished A5.4
PostCondition:    A7
Document access:  Paper-list(R/W), Conference-personDB(R), Conference-papers(R),
                  Conference-program(R/W), Session-descriptions(R/W),
                  Session-schedule(R/W)
```

A6 consists of two main sub-activities executed sequentially (A6.1 and A6.2):

A6.1: Suggest Sessions

The Program Chair is responsible for this activity, which can be decomposed further as follows:

- **A6.1.1 Match papers** Match all papers against keywords defined by the conference.
- **A6.1.2 Suggest sessions** Suggest a session division according to subjects and create a preliminary *Conference-program*.
- **A6.1.3 Set Up Session Committees** Choose a Session Chair (from PC Members) for every session.

A6.2: Select papers & Plan Sessions

The Session Chair is responsible for this activity. It can be decomposed further into the following six sub-activities:

- **A6.2.1 Determine session subject & goals** An initial *Session-description* will contain session subject and goals.
- **A6.2.2 Check papers for session** The Session Chairs should mark papers that are relevant for a session to notify their interest. The Session Chairs ID will be added to Possible-session-list in Paper-list for the selected papers.
- **A6.2.3 Paper allocation** If papers are marked by more than one Session Chair, these Session Chairs must negotiate (see N1 in 20.5.8) about which session is going to get the paper. Also if some papers are to marked by any Session Chair, Session Chair must be allocated. The final result from paper allocation to sessions will be updated in Actual-session in Paper-list. A6.2.3 will go on until all papers are allocated by one session.
- **A6.2.4 Check timeslot for session** Each Session Chair will mark timeslots for a session in Possible-session-list in Session-schedule.
- **A6.2.5 Session allocation** Sessions that have the same timeslot must negotiate (see N1 20.5.8). When all sessions are allocated, the result is added to Actual-session in the Session-schedule. Activity A6.2.5 will go on until all timeslots are allocated by one session.
- **A6.2.6 Publish session description** Each Session Chair will publish their *Session-description* to the Program Chair.

20.5.7 A7: Publish Conference Program

```
Function:        To finalise the conference program by combining
                 session descriptions.
Responsible:     PC Chair
PreCondition:    Finished A6.2.6
PostCondition:   None
Document access: Session-descriptions(R), Session-schedule(R), Conference-program(R/W),
                 Conference-personDB(R)
```

The final Conference-program is updated by adding the Session-descriptions from all sessions along with the the Session-schedule. The Conference-program will then be distributed to all involved and invited to the conference (registered in the Conference-personDB).

20.5.8 N1: Negotiation process

If more than three PC members are interested in reviewing the same paper (see A3.2 in section 20.5.3), or more than one Session Chair is interested in the same object (a paper in A6.2.3 or a timeslot in A6.2.5 in section 20.5.6), the following steps will be used to solve the conflict of selecting objects for the involved actors:

S1 The actor with *most objects marked*, will be forced to ”unmark” the conflicting object.

S2 All involved actors will be asked at one at a time to ”unmark” the conflicting objects.

S3 One or more involved actors will be picked out randomly to ”unmark” the conflicting object. This time the actor will be forced to ”unmark” object.

The process will start with S1. If S1 does not solve the conflict, S2 will be initiated. If still there is a conflict, S3 will be iterated until the conflict is solved.

We have developed the negotiation strategy described above, and it only shows *one approach* to solve the conflict in a simple and clean way. Other negotiation strategies could be used, but are not described here.

20.5.9 Instantiation of the scenario

This subsection describes how the conference organising process should be instantiated. The instantiation process involves allocation of human agents to roles, and making the scenario more concrete. The documents and the databases used in the scenario is described in appendix 20.5.10.

The conference organisation

The conference organisation consists of 25 PC Members. From these 25 PC Members, one person is selected to be PC Chair. The role PC Chair is responsible for managing the conference. In addition, five PC Members are selected to be Session Chairs. A Session Chair is responsible for arranging and managing a session. In our scenario, five sessions will be held according to the five Session Chairs selected.

Attendance and reviewing

To limit the possibilities for how the conference scenario should be modelled, a specific conference case called World Software Computing Conference 2000 (WSCC’00) will be used. At WSCC’00, 50 papers were received. All these papers were reviewed by three reviewers (meaning that a total of 150 reviews were performed). Every PC Member must review 6 papers in average. However, the PC Chair allows every reviewer to pick from four to eight papers to reviewed. If not all papers get three reviewers, reviewers with the fewest papers will be forced to review extra papers.

Acceptance of papers

As the proceedings for the conference want to have a certain quality on the papers accepted, only the 20 best (40 %) papers will be accepted. In activity A6.1.3 five Session

Chairs will be selected from the 25 PC Members. Each Session Chair will in the activity A6.2.2 mark interest for 3-5 papers to be allocated to their session.

Timeslots

The conference will run for four days, and there will be no parallel sessions on the conference. Every session (Session Chair) have to allocate two timeslots (four hours) from ten available timeslots for their session. The following timeslots are shown in the table below:

Day one	Day two	Day three	Day four
0900-1100 Timeslot1	0900-1100 Timeslot4	0900-1100 Timeslot7	0900-1100 Timeslot10
1200-1400 Timeslot2	1200-1400 Timeslot5	1200-1400 Timeslot8	1200-1500 Conference summary
1500-1700 Timeslot3	1500-1700 Timeslot6	1500-1700 Timeslot9	

20.5.10 Conference Documents and Databases

This section describes all documents and databases used in the scenario. It is also described how the documents and databases relates to specific activities.

Conference-personDB The **Conference-personDB** is a database containing potential participants to the conference as well as people in the program committee. A person is registered with: *PersonDB-number*, *Name*, *Address*, *Telephone-number*, *Fax-number*, *Email-address*, *Paper-number(s)*, *Role-information* [PC Chair, PC Member, Session Chair] , *Responses* [Letter of Intent, Late response], and *State-information* [Potential participant, Paper submitted, Paper accepted].
The *Paper number(s)* is used to as a key to refer to one paper or more papers in **Paper-list**.
The **Conference-personDB** is accessed by these activities: A1.2(R/W), A1.3(R), A2.1(R/W), A2.2A(R/W), A3(R), A5.4(R/W), and A6.1.3(R/W).

Paper-list The **Paper-list** is a list of all paper submitted where every entry has the following information: *Paper-number*, *Paper-title*, *PersonDB-number*, *Authors*, *Paper-keywords*, *Paper-location* (URL), *Possible-reviewer-list*, *Actual-reviewer-list*, *Possible-session-list*, *Actual-session*, *State* [Not-reviewed, Rejected, Uncertain, Accepted], and *Review-results*.
The *PersonDB-number* is used as a key to link the **Conference-personDB** and **Paper-list**. The *Possible-reviewer-list* is used to hold a list of *PersonDB-number* of PC Members interested in reviewing the paper (see activity A3.1). The *Actual-reviewer-list* hold a list of *PersonDB-number* to the three reviewers selected to review the paper. *Possible-session-list* and *Actual-session-list* are used accordingly to divide papers into sessions.
The following activities access **Paper-list**: A2.2(R/W), A3.1(R/W), A3.2(R/W), A4.1(R), A5.1(R), A5.4(R/W), A6.1.1(R), A6.2.2(R/W), and A6.2.3(R/W).

Session-schedule The **Session-schedule** is in this case a table containing ten *timeslots*. Every timeslot has two parameters: *Possible-session-list*, and *Actual-session*. *Possible-session-list* is used to list all sessions interested in one particular timeslot. *Actual-session* denotes that one session is selected for this timeslot.
These activities access **Session-schedule**: A6.2.4(R/W), A6.2.5(R/W), and A7(R).

Call-for-papers Initially a template for **Call-for-papers** exists.
These activities will access **Call-for-papers**: A1.1(R/W) and A1.3(R).

Acknowledge-templates These templates are used to give uniform responses from the program committee. Available templates are: Acknowledgement-for-received-paper, Acknowledgement-for-Letter-of-Intent, Regret-late-response, and Acknowledgement-of-paper-accept/reject.
These activities will use **Acknowledge-templates**: A2.1(R/W), A2.2(R/W), A2.3(R/W), A2.4(R/W), and A5.4(R/W).

Conference-papers These are papers received from authors. They are stored at a web-server with limited access to the PC Members. All Conference-papers have an unique URL. Additional paper information is stored in **Conference-personDB**.
These activities access **Conference-papers** (read-only): A2.2, A3.1, A4.1, A5.1, A5.2, A5.3, A6.1.1, and A6.2.2.

Review-reports Every paper review will result in a **Review-report** based on an available template. A **Review-report** is identified by a *PersonDB-number* and a *Paper-number*.
These activities access Review-reports: A4.2(R/W), and A5.1(R).

Session-description Each session writes **Session-description** reflecting the goal and content of a particular session.
These activities access **Session-description**: A6.2.1(W), A6.2.6(R/W), and A7(R).

Conference-program The **Conference-program** describes the contents of the conference, including all sessions.
These activities access **Conference-program**: A6.1.2(W), and A7(R/W).

20.5.11 Process Changes

This section describes the process changes that are going to be modelled and enacted in the three PCEs after the initial process has been modelled and enacted. Note that these process changes occur during enactment.

The following process changes should be applied to the scenario:

1. **Change activity sequence.** The activities A6.2.4 and A6.2.5 are exchanged with A6.2.2 and A6.2.3. The new sequence for A6.2 will then be A6.2.1, A6.2.4, A6.2.5, A6.2.2, A6.2.3, and A6.2.6 (see figure 20.7).

2. **Assign activity to another role when one role is not available.** The activity A2: Record response should be re-assigned to PC Member #1.

3. **Change negotiation strategy in N1.** The new negotiation strategy *N2*, involves three steps:

   S1 All actors that have selected conflicting objects, will be asked to ”un-select” these objects at the same time. The main difference with this this step and S2 in N1, is that this step is executed in parallel while S2 in N1 is executed sequentially.

   S2 For the remaining conflicting objects, a communication channel is opened between actors that have selected the same conflicting object. By using this communication channel (e.g. talk, IRC or similar), the actors can give arguments for who should ”unselect” this object. For paper reviewing, expertise within a certain research field can be used as an argument.

   S3 For the remaining conflicting objects, actors will picked out at random to ”un-select” these object until the conflict is solved.

4. **Change reviewer selection.** In a new version of the activity A3, this activity is monitored. If all reviewers are not allocated within a certain time, PC Chair will make a list of reviewers without any intervention from the PC Members.

20.6 Experimental Results

This section describes how the conference process was modelled in the three PCEs, and some results from the modelling.

20.6.1 Modelling used by all three environments

To make it easier to compare the different PCEs and to reduce the modelling time, we chose to represent all individual activities (opposed to cooperative activities) as web-pages. For the user, the individual activities are shown as web-pages describing what to do including HTML-forms, and have links to relevant tools and documents. All three PCEs accessed the same HTML-files for presenting the activities to the users. In appendix D, screenshots of the same activity (A1.1) is shown for the three PCEs as webpages (figure D.2 (Endeavors), figure D.4 (ProcessWeb), and figure D.6 (CAGIS PCE)). In Endeavors, a Java-based user-interface for explicitly notifying that the activity is finished is needed as shown in figure D.3. Endeavors also shows the animation of the process in activity networks as shown in figure D.1. In the CAGIS PCE, an agenda tool shows the agenda for users as shown in figure D.5. A tool for showing the agenda can also be implemented in ProcessWeb and Endeavors.

20.6.2 Modelling the Process in Endeavors

The modelling in Endeavors consisted of two main parts: The implementation of the handlers, and the graphical modelling of the process. A handler is executed when an activity in Endeavors is executed. We chose to implement the handlers in Java, since also the Endeavors framework is implemented in Java. The following activity handlers were implemented:

H1 **Show HTML handler** This handler was used to show HTML for individual web-pages. In addition, we used this handler to provide the cooperative activities Select paper (A3.1), Check paper for session (A6.2.2), and Check timeslot for session (A6.2.4). A3.1, A6.2.2 and A6.2.4 accessed a database through HTML-forms using CGI. The Java source for this handler is shown in appendix F in figure F.11.

H2 **Allocation handler** The allocation handler is responsible for detecting conflicting objects for current actor by accessing a database (Paper-list or Session-schedule) through a CGI. In addition, this handler will initiate the negotiation process (N1) as described in section 20.5.8. The Java source for an activity handler for paper allocation is shown in appendix F in figure F.12.

When the allocation handler is invoked, the user will be prompted with one of three possible user interfaces:

1. *No allocation conflicts*. This user interface will be shown if the user have selected no conflicting objects. This user interface is shown in figure E.1 in appendix D.
2. *User is forced to ”unselect object”*. This user interface is shown if S1 or S3 for current user is true (see negotiation process N1 in section 20.5.8). A screenshot of this user interface is shown in figure E.2.
3. *User is asked to ”unselect object”*. This user interface is prompted if S2 for current user is true. This user interface is shown in figure E.3.

20.6.3 Modelling the Process in ProcessWeb

The conference organising process was in ProcessWeb modelled in two parts. The first part modelled all individual activities, while the second part modelled cooperative activities.

Modelling the individual activities

The individual process was modelled as three roles: PC Chair, PC Member, and Session Chair. Each role has several states representing the activities that this particular role is responsible for. For each state, HTML was used to provide a user interface representing each activity. The role changes state when the user click on a button in an HTML-form, notifying ProcessWeb to change state. Interaction channels were implemented between roles, making it possible for e.g., notify the PC Member when PC Chair had finished an activity. The ProcessWeb PML source code is listed in appendix in G.1. In addition, a screenshot from ProcessWeb executing the activity A1.1 and A1.2 is shown in figure D.4. The reason for both A1.1 and A1.2 is shown in the same browser, is that these activities are executed in parallel.

Modelling the cooperative activities

Only the cooperative activities A6.2.4 and A6.2.5 were implemented in ProcessWeb. The reason for this was, that the other cooperative activities were similar and did not give any new challenges. The implementation of A6.2.4 and A6.2.5 was carried out by Mark Greenwood[4]. This was done to get an expert to implement the more complicated aspects of the scenario. In appendix G.2 the PML source for the activities A6.2.4 and A6.2.5 is listed. These activities have been modelled through two roles: SessionChair and TimeTable. In the SessionChair role, the user can select what timeslots (s)he is interested in. The TimeTable role is the role responsible for the allocation of timeslots for different sessions. From the TimeTable role, the allocation process is initiated, including the negotiation process N1 described in 20.5.8. In appendix E, figure E.4 shows a screenshot for the SessionChair role, and figure E.5 shows a screenshot for the TimeTable role.

20.6.4 Modelling the Process in CAGIS PCE

In CAGIS PCE, the scenario was modelled as CAGIS SimpleProcess PML, as a GlueModel and as cooperative agents.

CAGIS SimpleProcess model

The CAGIS SimpleProcess PML was used to model the individual activities in the conference organising scenario. All activities was described as $< activity >$ elements in an XML-document as shown in appendix H.1. Screenshots from demonstrating enactment of this process is shown in appendix D in figure D.5 and figure D.6. The former shows the agenda tool in SimpleProcess, while the latter shows the activity A1.1 enacted.

GlueModel

The GlueModel is used to model the relationships between the cooperative activities in CAGIS SimpleProcess model (previous paragraph) and the cooperative agents (next paragraph). The GlueModel contains six process fragment - agent pairs for the activities A3.1, A3.2, A6.2.2, A6.2.3, A6.2.4, and A6.2.5. In appendix H.2, the GlueModel for these activities are listed. For the selection activities (A3.1, A6.2.2, and A6.2.4), the GlueModel

[4]Mark Greenwood works for the the Informatics Process Group at University of Manchester

specifies that *selection agents* are used for selecting papers or timeslots. If the selection agent fails, current activity will be re-executed. The GlueModel further specifies that for the allocation activities (A3.2, A6.2.3, and A6.2.5), *negotiation agents* are initiated. If the allocation (of papers or timeslots) performed by the negotiation agents goes well, the next activity in SimpleProcess will be executed. If the negotiation process discovers that the user has to select more papers or timeslots, an activity for selecting papers or timeslots will be executed.

Cooperating agents

Cooperating agents are used in this scenario for selecting papers and timeslots, and for allocating the same resources. The agents used are mobile, and will move from a central agent meeting place (AMP) to workspaces for the users. The Java code for a selection agent is shown in appendix H.3.1, and the Java code for an allocation agent is shown in appendix H.3.2. When a selection agent visits a user in her/his workspace, the user-interface as shown in figure E.6 in appendix E is presented. Depending on the steps S1-S3 in the negotiation process N1 (see section 20.5.8), one of the following user interfaces will be prompted the user as shown in figure E.7, figure E.8, or figure E.9 respectively.
Cooperative agents is a separate research field, and we have done little on detailing cooperative/negotiation agents, and providing a rich infrastructure for such agents.

20.7 Evaluation of the three PCEs

In this section we will compare Endeavors, ProcessWeb and CAGIS PCE.

20.7.1 Coverage of the scenario

Table 20.2 shows results from measuring coverage and modelling time for modelling the conference scenario in the three PCEs. Note that additional time was used to create HTML for the individual activities used in all PCEs, and is not shown in the table.

The first research question we wanted to evaluate, was how complete the three different PCEs could model the conference organising scenario. For individual activities, all PCEs could completely model all activities. For cooperative activities (A3.1, A3.2, A6.2.2, A6.2.3, A6.2.4, and A6.2.5 [5]), we had some problems in modelling the allocation of papers and timeslots in Endeavors. The problem was how to represent negotiation process between different roles in an activity-network. We chose to solve this problem by invoking an allocation handler for each role at a time. After all roles had activated their allocation handler, the databases were checked to see if there were remaining paper or timeslots conflicts. If there were more conflicts, the same allocation handlers for all roles involved were executed again. This loop continued until all conflicts were solved. Figure F.4 shows

[5]The cooperative activity A5.3 was not modelled by any PCEs

graphical process model of A3: Reviewer selection, where a loop with A3.2 for all roles involved (here five) models the allocation process.

Modelling	**Endeavors**	**ProcessWeb**	**CAGIS PCE**
Individual activities	100 %	100 %	100 %
Cooperative activities	80 %	100 %	100 %
Modelling time for individual activities	6 hours	10 hours	3 hours
Modelling time for cooperative activities	10 days	2 days	5 days

Table 20.2: Coverage of scenario modelled

The modelling time in table 20.2 shows that for individual activities most time was used in ProcessWeb and least in CAGIS PCE, where Endeavors comes in between. One explanation could be that the modeller was most familiar with CAGIS PCE. However modelling the individual activities in Endeavors and ProcessWeb was very straight forward, and we were not delayed by any time consuming problems. The difference in modelling time can be explained as following:

- **ProcessWeb**. The PML in ProcessWeb is close to a textual object-oriented programming language. This means that the modeller needs to “program” the process as roles with different states representing the activities the role is responsible. In addition, the modeller has implement the infrastructure for interaction between roles, role assignments, and role configuration. The main reason more time had to be used to model in ProcessWeb, was that the model had to be “programmed” and addition configuration and infrastructure had to be modelled.
- **Endeavors** More time was spent on modelling in Endeavors compared to CAGIS PCE, because in Endeavors we had to implement an activity handler that initiated an HTML-document in a Web-browser. If we don’t consider the time spent on implementing the activity handler, Endeavors was more efficient for modelling the individual activities than CAGIS PCE.

According to table 20.2, ProcessWeb is the most efficient PCE for modelling cooperative activities. The reason for this is that the PML used in ProcessWeb is designed to efficiently implement role-interaction. CAGIS PCE used an agent-API to implement the cooperative activities. The current version of our CAGIS agent-API is too low-level for efficient cooperative activity modelling. It should also be noted, that the user interface used in the CAGIS PCE was more advanced compared to simple web-interface in ProcessWeb. In Endeavors the cooperative activities was implemented in Java from scratch, making this the least efficient approach. Richard N. Taylor at UCI[6] has suggested to use globally visible blackboard for papers and timeslots in Endeavors, accessed by various roles. This was

[6]Professor Richard N. Taylor works for Department of Information and Computer Science, University of California, Irvine

not implemented because the documentation for implementing such blackboards were not available.

20.7.2 Adaptability of Process Change

This subsection describes how efficient the three PCE can adopt to the process changes described in section 20.5.11. In all of Endeavors, ProcessWeb and CAGIS PCE, the process model can be changed during enactment. In Endeavors, the process model can graphically be manipulated during enactment, and activity handlers can dynamically be changed as well. In ProcessWeb, a process change is provided by extracting data and state from a current role, a new role definition replaces the current role, and the extracted data and state is restored in the new role definition. The CAGIS PCE, the activity network can be changed by moving process fragments to another workspace or another place in the workspace.

In table 20.3, the ability measures are mapped to different process changes to the three PCEs. The first two process changes were implemented while the last two were not. For the latter, we have investigated how to implement these process changed and we have estimated how much effort these change requires. The result reflects how much effort must be spent to implemented the change, and is scaled 1-5 (where 5 is *Very little effort* and 1 is *Very strong effort*):

Process change	**Endeavors**	**ProcessWeb**	**CAGIS PCE**
1. Change activity sequence	5	3	5
2. Assign activity to another role	5	2	5
3. Change negotiation strategy	1	3	5
4. Change reviewer selection	1	3	5

Table 20.3: PCE adaptability to Process changes

Here are comments on results for each process change (1-4) shown in table 20.3:

1. **Change the activity sequence** In Endeavors, to do this process change, you simply have to remove two control flow arcs and add two new ones. In ProcessWeb this particular process change depends on how the activities are represented in the roles. If the activities are represented as states in the role, some if-sentences must be changes to change the activity sequence. The changed role definition must in addition be compiled into the system, and the role must be modified to get the new behaviour. In CAGIS PCE, to change an activity sequence, you simply state where you want to move the activities (the process fragment).

2. **Assign an activity to another role** In Endeavors role assignment can be changed by editing the *AssignedTo* attribute for an activity through Endeavors graphical user

interface. In ProcessWeb assigning an activity to another role is more complicated. First, the PML code describing the activity must be transfered from one role to another. Second, the PML code for both roles (source and target roles) must be changed to cope with the removal of activity in the source role and adding the activity in the target role. Third, PML code for both roles must be compiled, and both roles must be modified to their new behaviour. The reason it is hard to assign an activity to another role in ProcessWeb, is that the role is the unit of change. This problem can be avoided by modelling all activities as separate roles. In CAGIS PCE, hierarchical workspaces are used to represent roles. To re-assign an activity to another role, this activity is simply moved to another workspace.

3. **Change negotiation strategy** We have not considered the effort of implementing the new negotiation strategy, but rather how to integrate an already implemented negotiation strategy into the process. In Endeavors, it is rather hard to implement the N2 negotiation strategy (described in section 20.5.11). This is because it is hard to represent a network of interacting roles using an activity-based process formalism. The new negotiation strategy demands communication across different roles, and is hard to represent in an activity network. ProcessWeb is probably the best environment for implementing the new negotiation strategy, but to incorporate this strategy with existing roles can be hard. N2 demands extensive changed in the roles that can make it hard to migrate from an old role definition to new one. If this process change was known in advance, the process model could be implemented to handle such massive changes. It should be noted this problem has been addressed in ProcessWeb through support for meta-process. A process architecture is used to provide support for a generic change making migration easier. Essentially making one change is hard, but with a meta-process you can spread this cost over many changes over the lifetime of your system. In CAGIS PCE, it is very easy to change negotiation strategy by editing the GlueModel for the process fragments involving negotiation agents. To implement the N2 in the agent system is probably more time consuming than using ProcessWeb, but it is really easy to change negotiation strategy if a matching implementation (cooperative pattern) is already available.

4. **Change reviewer selection** This process change demands that the activity A3: Reviewer selection is monitored and if three reviewer are not allocated to all papers within a certain time, PC Chair will do the reviewer selection. In Endeavors this process change can be implemented by adding timeout functionality for the activity handlers in A3. When the timeout event occur, a control activity can be used to route the workflow to PC Chair that have to do the reviewer selection on her/his own. This process change demands a lot of changes in both the activity handlers and the activity network. In ProcessWeb this process change is easier to implement, but will also demand some effort. The role assigned originally for A3 (PC Member) need to have a timeout function for selecting and allocating reviewer to papers. When this timeout function is triggered, a message is sent to the PC Chair role that (s)he must do the reviewer to paper allocation. This means that the PC Chair and the PC Member role definitions must be changed and the communication channel

between them must be configured. In CAGIS PCE the GlueModel can be used to implement this process change. A monitor agent will monitor the reviewer allocation agents and if they are not finished within a certain time, the monitor agent will notify the GlueServer. The GlueServer will then initiate an execution of a Reviewer selection process fragment for PC Chair. The definition of the Reviewer selection process fragment for PC Chair must be defined. Since monitor agents is a part of the CAGIS DIAS, changes must be made in the GlueModel, and an additional process fragment for reviewer selection for PC Chair must be defined.

20.7.3 Reflection on the Evaluation

Looking back on the research questions in section 20.3.2, we can from the results in the two above sections say that a combination of a traditional workflow system and a software agent system is better compared to a stand-alone workflow system to model and enact cooperative processes and adapt to process changes. Some objections can however be raised to this statement:

O1 **The selection of scenario is biased.** The reason for selecting the conference organising scenario was in order to use a scenario that was already described in the literature (external validity). One important question is to ask whether the conclusion of this evaluation is valid for other scenarios describing cooperative processes. To answer this question, we have to look at how the conference scenario has been modelled. We have distinguished between individual and cooperative activities. Individual activities are activities where one role is assigned for performing this activity. In cooperative activities, several roles are involved in performing the activity. We believe that as long as other scenarios are modelled by distinguishing between individual and cooperate activities, the evaluation result will be valid. If another approach is used to model the process, the evaluation result is not necessary valid.

O2 **More experience with your own environment.** A problem with our experiment, is that we are more experienced with our own environment than the others. This means that the modelling time can be unreliable. To address this problem, we spent about one week to exercise process modelling in Endeavors and about one month with modelling process in ProcessWeb. The reason more time was spent on ProcessWeb, was that the author had used ProcessWeb for modelling before the experiment was planned. In addition, we had no time-consuming problems when modelling the individual activities in Endeavors and ProcessWeb. For cooperative activities a modelling expert in ProcessWeb was used, but not for Endeavors. This means that the modelling time for cooperative activities activities is likely to be less if an expert had been used. However, since modelling interaction between roles in Endeavors is not a part of the process modelling language, it is likely that the evaluation result will be the same.

O3 **Statistical invalid data.** Using only one target process (conference process) we can not statistically validate our two research questions only based on the quantitative data we have collected. For a statistically valid experiment, data should be collected from several scenarios modelled in the three PCEs. The collected data has only been used as an indication of the evaluation result, and qualitative discussions have enlighten the results from data collection.

O4 **The selection of process changes is biased.** We have tried to pick out process changes that are likely to occur in the conference scenario. It is possible that our selection is to limited, and more changes should be considered. Other process changes could have indicated a more nuanced score for the three PCEs. However, we believe that the selected process changes represent a wide spectrum of possible changes making the result believable.

It should be noted that Endeavors and ProcessWeb are much more mature environments compared to CAGIS PCE, and they offer richer semantics for expressing the process. In CAGIS PCE, we have used agents to provide the functionality that we lose through having only a simple workflow model. Since the agent-API is still to low-level, more advanced workflow can be time consuming to implement in CAGIS PCE.

20.8 Conclusion

In this report we have endeavoured to investigate if a combined workflow - agent system is better able to model cooperative processes and to implement to process changes compared to a stand-alone workflow system. We have modelled a conference organising scenario in the PCEs Endeavors, ProcessWeb, and CAGIS PCE, and collecting some data during the modelling of the scenario. Our collected data and discussions indicate that the combined approach performs very well in respect to cooperative processes and process changes. This result is an encouragement to continue our research.

This evaluation was mainly performed by the author. For similar future evaluations, we would like to avoid this approach to ensure the validity of the results. One approach could be to use students that are not familiar with any of the PCEs, give them a proper introduction to the PCEs (the same amount), and let them model some processes. By using this approach, we would be able to collect more data, and could make some statistical analysis. A problem would be to find and pick descriptions of neutral scenarios that not favour one of the PCEs. Another approach would be to use experts on each PCE to model processes. A problem with this approach is to deal with interpretations of the scenarios. Also if the experiment was not controlled, it could be possible to fake the measurements. We can conclude that doing such experiments are really hard and time-consuming.

The main contribution of our work is the GlueServer, making it possible to define couplings between activity-based workflow and software agents. With regards to efficiently, ProcessWeb was the best environment to model cooperative activities. When the activity handlers had been implemented in Endeavors, Endeavors was most efficient for modelling individual activities. Future work should therefore investigate how a combination of Endeavors, ProcessWeb and CAGIS GlueServer would work (federation of workflow systems). This combination features solid and rich modelling support for both individual and cooperative activities, where the CAGIS GlueServer acts as a middleware. Further, the GlueServer can be used to combine more than two workflow systems, allowing loosely coupled, autonomous entities choose their own workflow tool. In this way, the CAGIS PCE provides cooperative support for a heterogeneous environment. Current implementation for the CAGIS GlueServer does not yet provide a federation of workflow systems through Workflow Management Coalition's interoperability workflow-XML binding. Other APIs should also be considered.

Acknowledgements

First we would like to thank Mark Greenwood for modelling parts of the conference management process in ProcessWeb, and for giving valuable feedback during the experiment. We would also like to thank Reidar Conradi, Ian Robertson, Richard N. Taylor, and Letizia Jaccheri for useful comments.

Part IV

Appendix

APPENDIX A

CAGIS DIAS Design Specifications

Here is a summary of the design specifications of the CAGIS Distributed Intelligent Agent System (DIAS) extracted from the technical report [PHBN99, HN00, SW00].

A.1 Agent Design Specifications

This subsection describes the design specifications for the different types of agents in the DIAS. There are three main categories of agents; *system agents*, *participation agents*, and *user agents*.

A.1.1 System Agents

System agents administrate the Agent Meeting Places (AMPs) in the DIAS II architecture. All system agents are stationary agents.

Manager Agents

The manager agent is responsible for managing the AMPs, and need to correspond to the following design specifications:

```
[D1]   Enable creation and deletion of AMPs.
[D2]   Enable user clients to connect to an AMP.
[D3]   Register new agents in AMPs.
```

```
[D4]   Clean up when an agent has left an AMP.
[D5]   Advertise an AMP to other AMPs.
[D6]   Register information about other AMPs.
```

Facilitator Agents

These agents play an important role when agents communicate, and need to correspond to the following design specification:

```
[D7]   Find a suitable receiver of a message when the receiver ID is
       not specified.
```

Monitor Agents

The monitor agents are logging events in AMPs and watching security. A monitor agent must:

```
[D8]   Log agents when they arrive and leave an AMP.
[D9]   Log messages sent between agents when they are sent via an
       facilitator agent.
[D10]  Make sure the security policy is upheld.
```

Repository Agents

These agents access repositories and must correspond to the following design specifications:

```
[D11]  Retrieve required information in repositories on behalf of a
       user or other agents.
[D12]  Insert information in repositories on behalf of a user or
       other agents.
```

Mediation Agents

These agents work for an agreement between negotiation agents. Meditation agents should:

```
[D13]  Force an agreement between negotiating agents if the
       negotiation has taken too much time or used too many
       interactions (specified by an administrator). Mediation is
       performed on behalf of a negotiation agent or predefined rules
       found in a repository.
```

Interface Agents

These agents are connected to the ORB and must correspond to the following design specifications:

```
[D14]  Connected to the ORB.
[D15]  Implement the MASIF interfaces.
[D16]  Provide an interface for interaction with agents in the DIAS
       system.
[D17]  Registered in the Implementation Repository of the ORB. This
       means that the Interface Agent must know which server classes
       that are supported on the server. The ORB uses this
       information to locate active objects.
```

A.1.2 Participation Agents

Participation agents can either be stationary or mobile, and facilitates communication between agents.

Communication Agents

These agents facilitates communication between agents, and must correspond to the following design specifications:

```
[D18]  Bring a message from one agent to another when the agents are
       situated in different AMPs.
[D19]  Provide a mechanism for telling the sender of the message
       whether the messages was received or not, or whether some
       error occurred during sending.
```

Negotiation Agents

Negotiation agents provide help for a user or other agents to reach an agreement, and must correspond to the following design specifications:

```
[D20]  Negotiate on behalf of the user or other agents
[D21]  Negotiate within a number of interaction or within a
       predefined time quantum.
```

```
[D22]  Perform all negotiations in an AMP.
[D23]  Negotiation agents can be interrupted by a mediating agent if
       NA.2 fails.
```

KQML Agents

A KQML agent enables the user to directly send messages from a user to agents in DIAS, and must correspond to the following design specifications:

```
[D24]  Provide a simple GUI for entering KQML performatives and
       parameters.
[D25]  One KQML agent should always be available at an AMP.
```

A.1.3 User Agents

User agents are agents that are created to interact with the user. Such agents can either be mobile or stationary, and must correspond to the following design specifications:

```
[D26]  Must conform the the DIAS developer API.
[D27]  Must be either mobile or stationary.
```

A.2 Agent Meeting Place Design Specifications

The Agent Meeting Place (AMP) is where agents interact and exchange messages and services. An AMP must correspond to the following design specifications:

```
[D28]  An agent should be able to register itself with the following
       parameters according to the MASIF standard: Agent name, Agent
       location, Agent system type, Language id, Agent description,
       Major version, Minor version, and Agent properties.
[D29]  Receive agents and allow the to reside in the AMP.
[D30]  Enable possible rejection of agents based on security
       violations.
[D31]  Enable pausing of an agent by saving the agent to disk, and
       later resume it.
[D32]  Facilitate at least a facilitator agent, a manager agent, and
       a monitor agent.
[D33]  An AMP should be described according to the MASIF standard:
       Agent place name, System type, Language map, Agent place
       description, Agent place description, Major version, Minor
       version, and Agent place properties.
[D34]  Facilitate logging of events that takes place in the AMP.
```

```
[D35]  Facilitate logging of messages sent to agents via the
       facilitator agent.
[D36]  Provide an administrator interface for removing agents.
[D37]  Facilitate negotiation using negotiation agents in the AMP.
[D38]  Facilitate mediation between negotiating agents according to
       some specified parameters.
[D39]  Inform other AMPs about its services.
[D40]  Store information about other AMPs.
```

A.3 Agent Interface Design Specifications

Here is a list of design specifications that must be fulfilled to enable the agent system to interact with users and other systems:

```
[D41]  Interaction with other agent systems should be provided using
       an Object Request Broker (ORB) with MASIF interface objects.
[D42]  Agent system user clients enable the user to subscribe to AMPs
       from her/his workspace.
[D43]  Agent system user clients enable the user to initiate,
       configure, interact, and terminate her/his agents.
```

APPENDIX B

CAGIS SimpleProcess Design Specifications

CAGIS SimpleProcess is a simple workflow system providing the users with a web-interface to interact with the system.

B.1 CAGIS SimpleProcess Architecture Design Specifications

The architecture of CAGIS SimpleProcess should correspond to the following design specifications:

```
[D44]  A process model can be a collection of coupled process model
       fragments that can be distributed over several workspaces.
[D45]  A process fragment can be moved from one workspace to another.
[D46]  A process fragment definition can be edited locally in a
       workspace.
[D47]  Process fragments can be distributed on different sites.
```

B.2 CAGIS SimpleProcess PML Design Specifications

The CAGIS SimpleProcess Process Modelling Language (PML) should fulfil the following design specifications:

```
[D48]  A process model is described in XML.
[D49]  A process has a name and can consist of one or more
       process fragments.
[D50]  A process fragment has a name and is associated with a
       workspace, and consists of one or more activities.
[D51]  An activity has a name, a state, a due time, a description, a
       code, and is associated with a workspace.
[D52]  Activities are linked by prelinks and postlinks, similar to
       hyper-links. A prelink holds an identifier to an activity prior
       to current activity. A postlink holds an identifier to an
       activity that is going to be executed after current activity.
[D53]  Feedback loops are allowed in activity networks.
[D54]  An activity can have three states: Waiting, Ready, and Finish.
[D55]  The code is used to hold HTML-code or an reference to a
       HTML-file.
```

B.3 CAGIS SimpleProcess Tool Design Specifications

The CAGIS SimpleProcess tools should fulfil the following design specifications:

```
[D56]  The CAGIS SimpleProcess tools should provide a interface
       available through a standard web-browser.
[D57]  The process server should manage the state information of
       activities.
[D58]  The process server should enable registration and initiation
       of new activities.
[D59]  The process server should enable removal of activities.
[D60]  The process server should enable moving activities from one
       workspace to another, and from one process server to another.
[D61]  The process modeller should provide a simple HTML-interface
       for entering, changing, removing and browsing activities.
[D62]  The agenda manager should present agenda information for all
       workspaces, and activate activities that are ready to be
       executed.
[D63]  The agenda manager should provide an interface for navigating
       through the activity-network enabling users to explore
       activity dependencies.
[D64]  The monitor tool}, should enable an interface to monitor state
       and progress of the process.
```

APPENDIX C

CAGIS GlueServer Design Specifications

The GlueServer is a middleware used to facilitate interaction between software agents (CAGIS DIAS) and a workflow system (CAGIS SimpleProcess). Here is a summary of design specifications for the GlueServer based on the design specifications described in [Bjø00].

C.1 GlueServer Design Specifications

The GlueServer should correspond to the the following design specifications:

```
[D65]  Enable an interface to CAGIS DIAS through a DIAS interface
       agent.
[D66]  Use CORBA to communicate to other systems (through MASIF).
[D67]  Enable an interface to CAGIS SimpleProcess through a CGI
       interface.
[D68]  Enable to receive and process communication requests from
       process fragments.
[D69]  Be able to read GlueModels from some repository, and store it
       as an internal structure.
[D70]  Be able to find the correct process fragment - agent pair
       based on information received from the workflow tool.
[D71]  Execute the following reactions: Execute process fragment,
       Move process fragment, Halt process fragment, Change and
       re-execute process fragment, Add new process fragments, Remove
       process fragment, Start new interaction by agent, Stop
       interaction by agent, Create a new Agent Meeting Place, and
       Remove Agent Meeting Place.
```

```
[D72]  Provide a general interface to mobile agent systems defined by
       OMG’s MASIF standard.
[D73]  Provide a general workflow interface using the
       interoperability workflow-XML binding framework.
```

C.2 GlueModel Design Specifications

The GlueModel should fulfil the following design specifications:

```
[D74]  The GlueModel should be available as an XML-document.
[D75]  The GlueModel should be according to the formalism below:

        FRAGMENT-AGENT-PAIR
          AGENT
            (key AGENT-CLASS,
             in AMP-ID,
             in INTERACTOR+,
             INTERACTION-TYPE,
             RESULT)
          PROCESS FRAGMENT
            (key FRAGMENT-ID,
             <RESULT : REACTION>*)
```

APPENDIX D

Screenshots from enacting Individual activities

This appendix shows screenshots of the user-interfaces for individual activities (not cooperative) in the conference scenario for the three different process centred environments; CAGIS PCE, Endeavors, and ProcessWeb. The screenshots from the different PCEs are quite similar, because all user interaction is provided through HTML and webpages.

In the figures D.2, D.4, and D.6 the activity “A1.1 Make call for papers” is shown for Endeavors, ProcessWeb, and CAGIS SimpleProcess respectively. The main difference between the way the activities are shown, is that CAGIS SimpleProcess and ProcessWeb have additional interface for notifying the workflow system that the user is finished with the activity A1.1. In Endeavors, an additional Java user-interface is used to notify the workflow system that an activity is finished, as shown in figure D.3. CAGIS SimpleProcess also provides an agenda for each role in the process as shown in figure D.5. This feature can also be implemented in Endeavors and ProcessWeb. Figure D.1 shows how Endeavors animate the execution of the process.

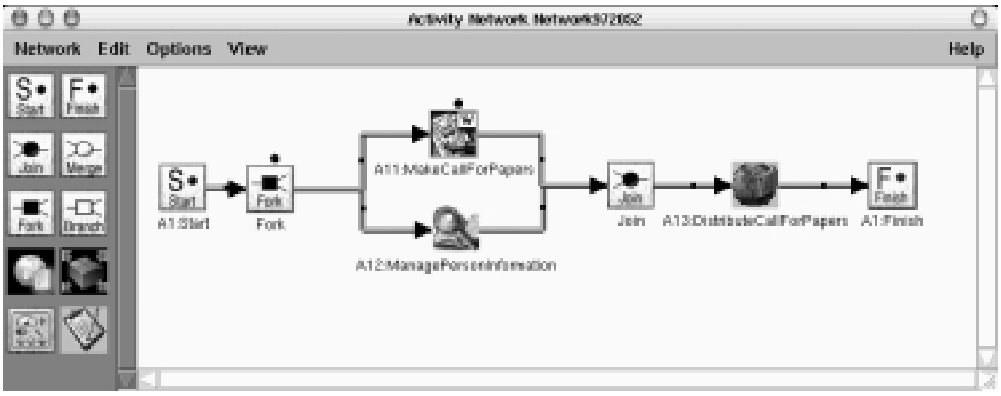
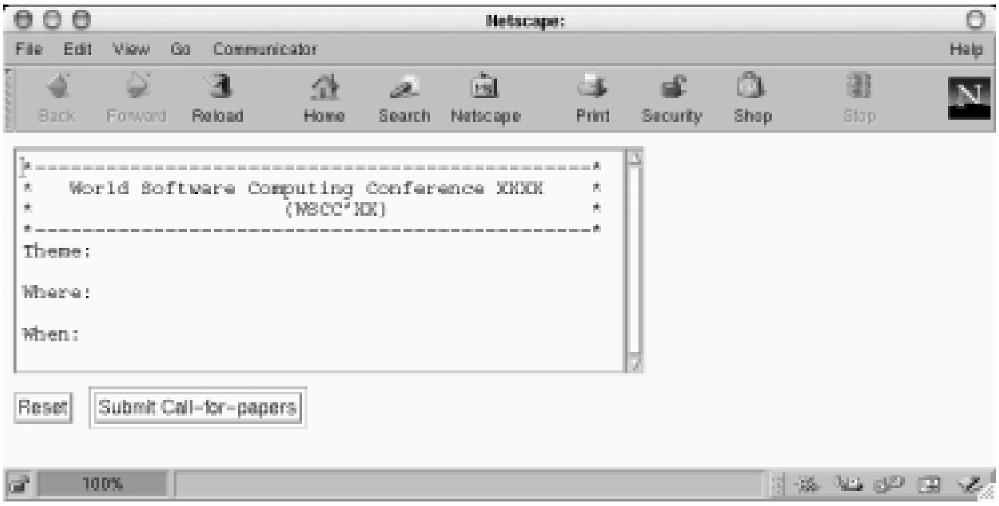


Figure D.1: Execution of the activity A1.1 in Endeavors



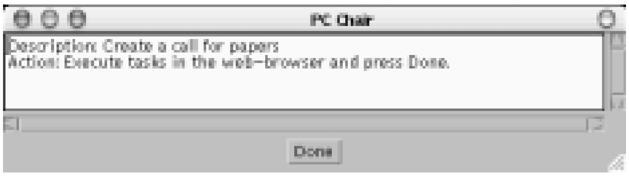
```
*-----------------------------------------------*
*    World Software Computing Conference XXXX    *
*                 (WSCC'XX)                      *
*-----------------------------------------------*
Theme:

Where:

When:
```

Reset
Submit Call-for-papers

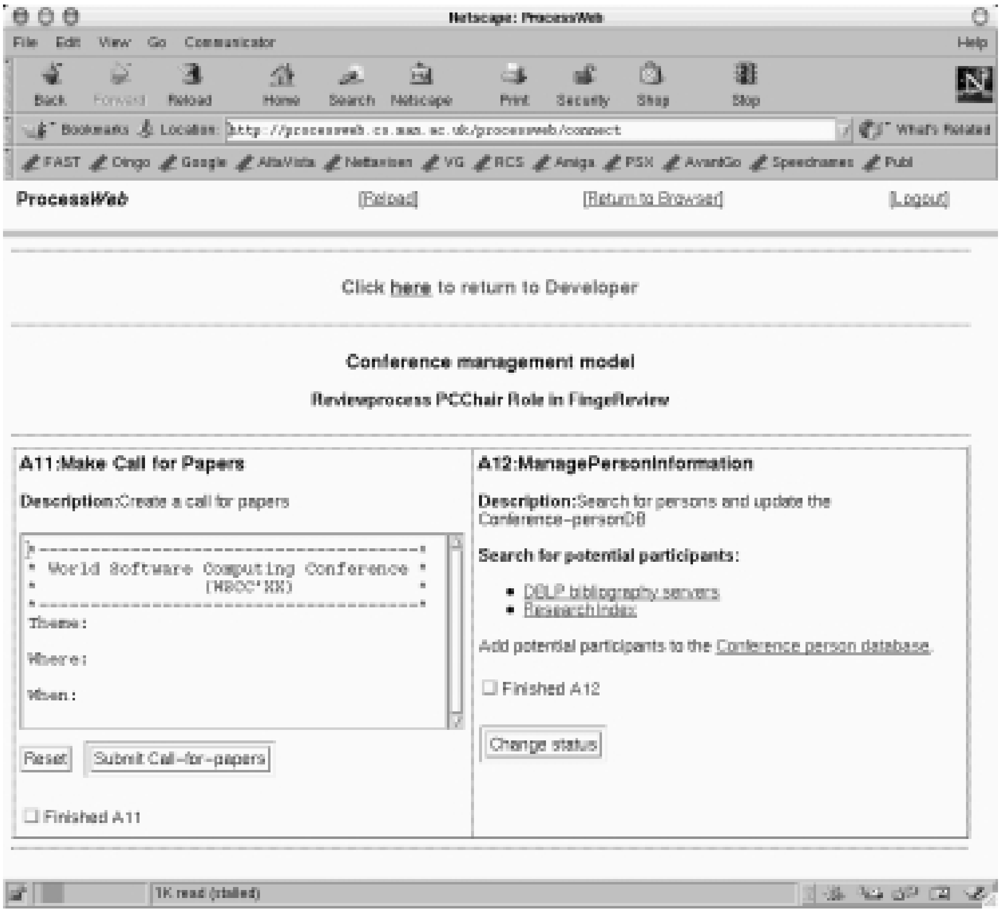
Figure D.2: Activity A1.1 shown in Endeavors



Description: Create a call for papers
Action: Execute tasks in the web-browser and press Done.

Done

Figure D.3: Interface to notify Endeavors that an activity in finished


Netscape: ProcessWeb
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://processweb.cs.man.ac.uk/processweb/connect What's Related
FAST Origo Google AltaVista Nettavisen VG RCS Amiga PSX AvantGo Speednames Publ
1K read (stalled)

ProcessWeb [Reload] [Return to Browser] [Logout]

Click here to return to Developer

Conference management model

Reviewprocess PCChair Role in FingeReview

A11:Make Call for Papers

Description: Create a call for papers

```
*-------------------------------------*
* World Software Computing Conference *
*             (WSCC'XX)               *
*-------------------------------------*
Theme:

Where:

When:
```

Reset | Submit Call-for-papers

☐ Finished A11

A12:ManagePersonInformation

Description: Search for persons and update the Conference-personDB

Search for potential participants:

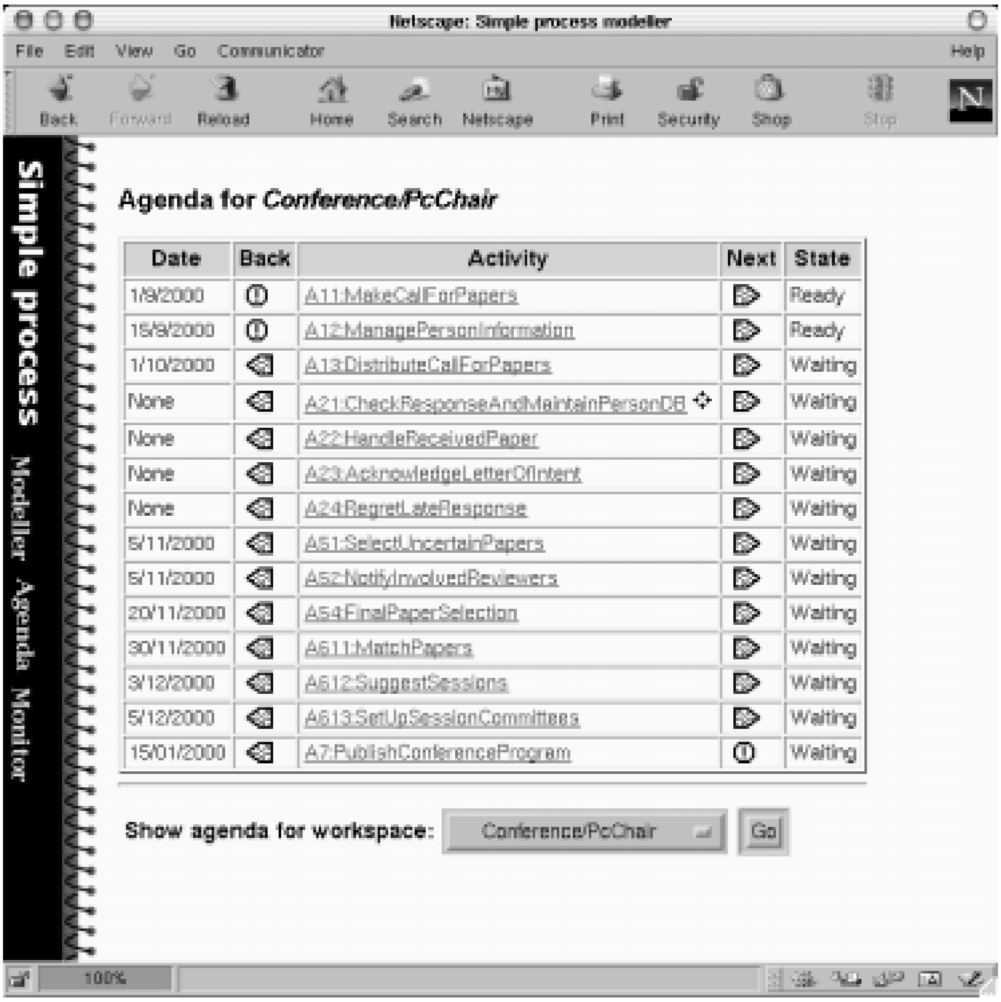
- DBLP bibliography servers
- ResearchIndex

Add potential participants to the Conference person database.

☐ Finished A12

Change status

Figure D.4: Activity A1.1 and A1.2 shown in ProcessWeb

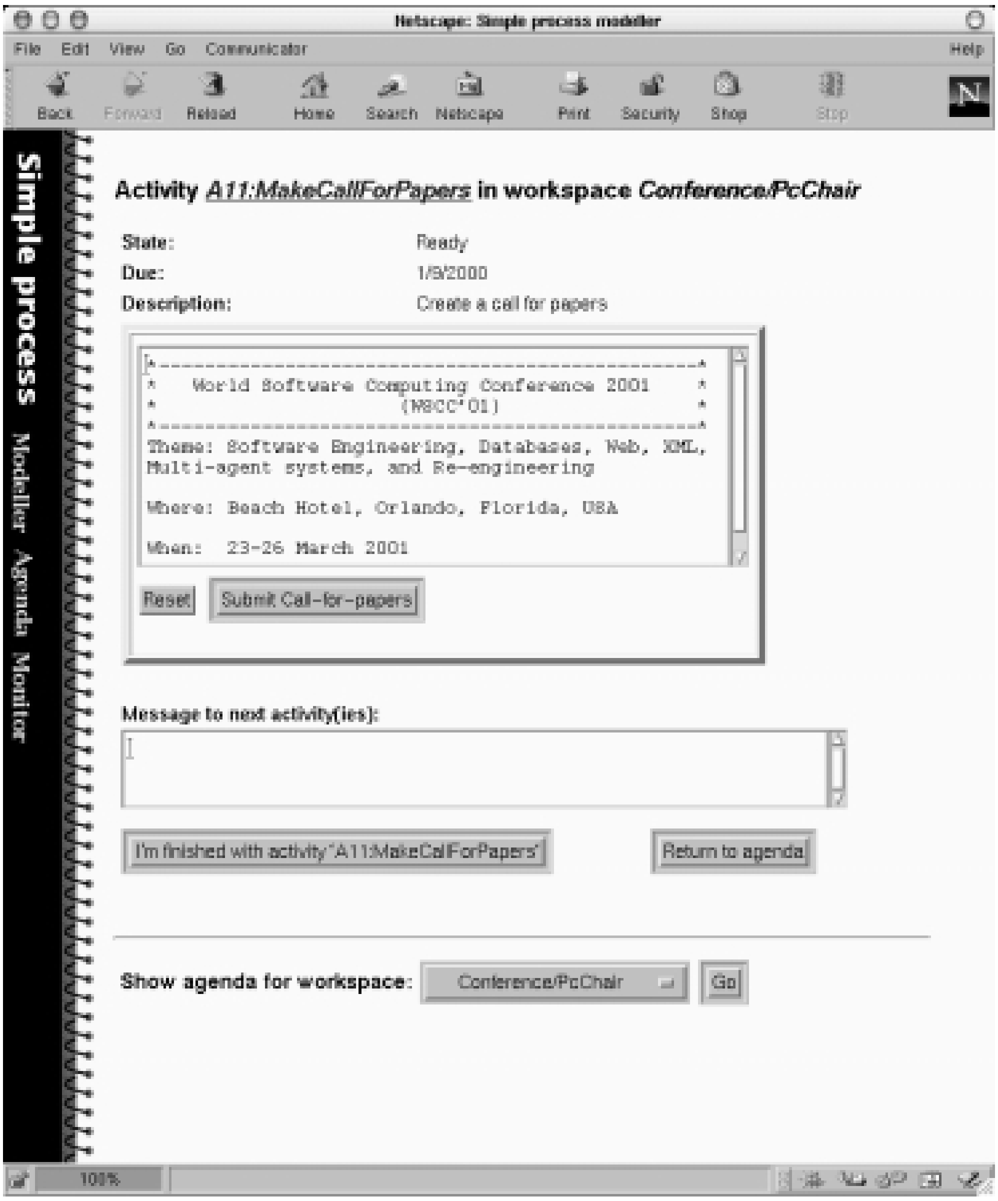

Netscape: Simple process modeller
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Simple process
Modeller
Agenda
Monitor
100%

Agenda for *Conference/PcChair*

Date	Back	Activity	Next	State
1/9/2000	⊙	A11:MakeCallForPapers	▷	Ready
15/9/2000	⊙	A12:ManagePersonInformation	▷	Ready
1/10/2000	◁	A13:DistributeCallForPapers	▷	Waiting
None	◁	A21:CheckResponseAndMaintainPersonDB	▷	Waiting
None	◁	A22:HandleReceivedPaper	▷	Waiting
None	◁	A23:AcknowledgeLetterOfIntent	▷	Waiting
None	◁	A24:RegretLateResponse	▷	Waiting
5/11/2000	◁	A51:SelectUncertainPapers	▷	Waiting
5/11/2000	◁	A52:NotifyInvolvedReviewers	▷	Waiting
20/11/2000	◁	A54:FinalPaperSelection	▷	Waiting
30/11/2000	◁	A611:MatchPapers	▷	Waiting
3/12/2000	◁	A612:SuggestSessions	▷	Waiting
5/12/2000	◁	A613:SetUpSessionCommittees	▷	Waiting
15/01/2000	◁	A7:PublishConferenceProgram	⊙	Waiting

Show agenda for workspace: Conference/PcChair [Go]

Figure D.5: Agenda for PC Chair in CAGIS SimpleProcess


Netscape: Simple process modeller
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Simple process
Modeller Agenda Monitor
Reset
Submit Call-for-papers
I'm finished with activity 'A11:MakeCallForPapers'
Return to agenda
Go
100%

Activity *A11:MakeCallForPapers* in workspace *Conference/PcChair*

State:	Ready
Due:	1/9/2000
Description:	Create a call for papers

```
*------------------------------------------------*
*   World Software Computing Conference 2001     *
*                   (WSCC'01)                    *
*------------------------------------------------*
Theme: Software Engineering, Databases, Web, XML,
Multi-agent systems, and Re-engineering

Where: Beach Hotel, Orlando, Florida, USA

When:  23-26 March 2001
```

Message to next activity(ies):

Show agenda for workspace: Conference/PcChair

Figure D.6: The activity A1.1 in CAGIS SimpleProcess

APPENDIX E

Screenshots from enacting Cooperative Activities

This appendix shows screenshots taken from enactment of cooperative activities in Endeavors, ProcessWeb and CAGIS PCE.

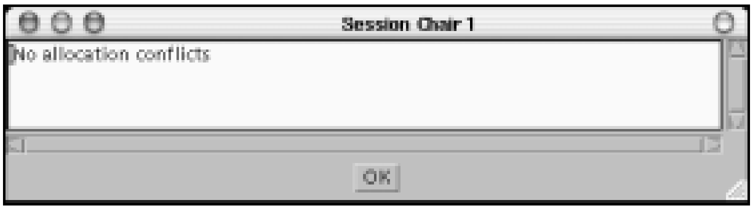
In Endeavors, one of three user interfaces can be prompted when allocating an object (paper or timeslot):

- O1 When there is no allocation conflict, the user interface in figure E.1 is shown.
- O2 When a user is forced to “unselect” an object, the user interface in figure E.2 is shown.
- O3 When a user is asked to “unselect an object, the user interface in figure E.3 is shown.

These user interfaces are taken from the cooperative activity *A6.2.3: Paper allocation*.

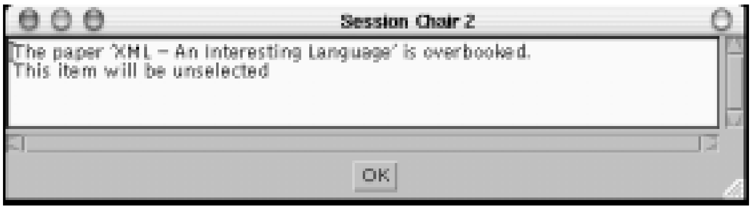
In ProcessWeb, screenshots from the cooperative activities *A6.2.4: Check timeslot for session* and *A6.2.5: Session allocation* have been taken. In figure E.4, we can see the user interface used to request timeslots for Session Chair 1. In figure E.5, a screenshot of the role *Timetable* is shown. The Timetable role is responsible for allocating timeslots to sessions through negotiation steps S1-S3.

In CAGIS PCE, we have included screenshots from the cooperative activities *A3.1: Select paper*, and *A3.2: Reviewer allocation*. Figure E.6 shows a screenshot from a **Paper selection agent** (for activity A3.1). Figures E.7, E.8, and E.9 shows screenshots from the **Reviewer Allocation Agent** in action.



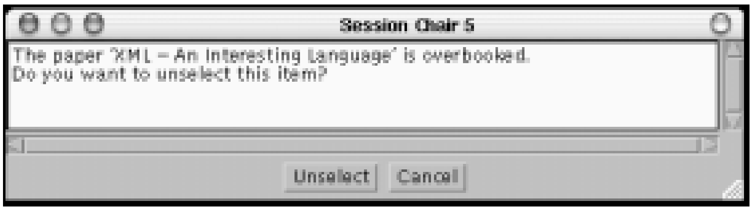
No allocation conflicts

Figure E.1: No allocation conflicts for Session Chair 1 in Endeavors



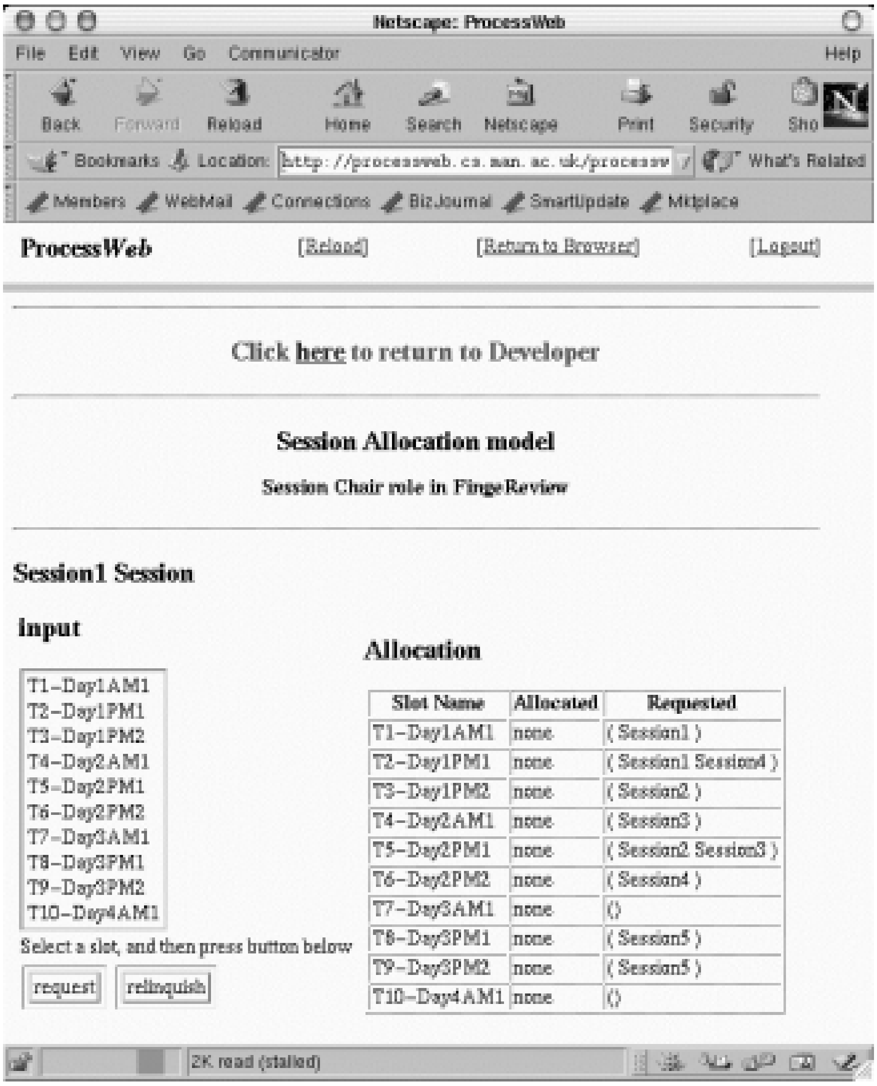
The paper 'XML – An Interesting Language' is overbooked.
This item will be unselected

Figure E.2: Session Chair 2 is forced to "unselect" a paper in Endeavors



The paper 'XML – An Interesting Language' is overbooked.
Do you want to unselect this item?

Figure E.3: Session Chair 5 is asked to "unselect" a paper in Endeavors


Netscape: ProcessWeb
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Sho
Bookmarks Location: http://processweb.cs.man.ac.uk/processw What's Related
Members WebMail Connections BizJournal SmartUpdate Mktplace
2K read (stalled)

ProcessWeb [Reload] [Return to Browser] [Logout]

Click here to return to Developer

Session Allocation model

Session Chair role in FingeReview

Session1 Session

Input

T1-Day1AM1
T2-Day1PM1
T3-Day1PM2
T4-Day2AM1
T5-Day2PM1
T6-Day2PM2
T7-Day3AM1
T8-Day3PM1
T9-Day3PM2
T10-Day4AM1

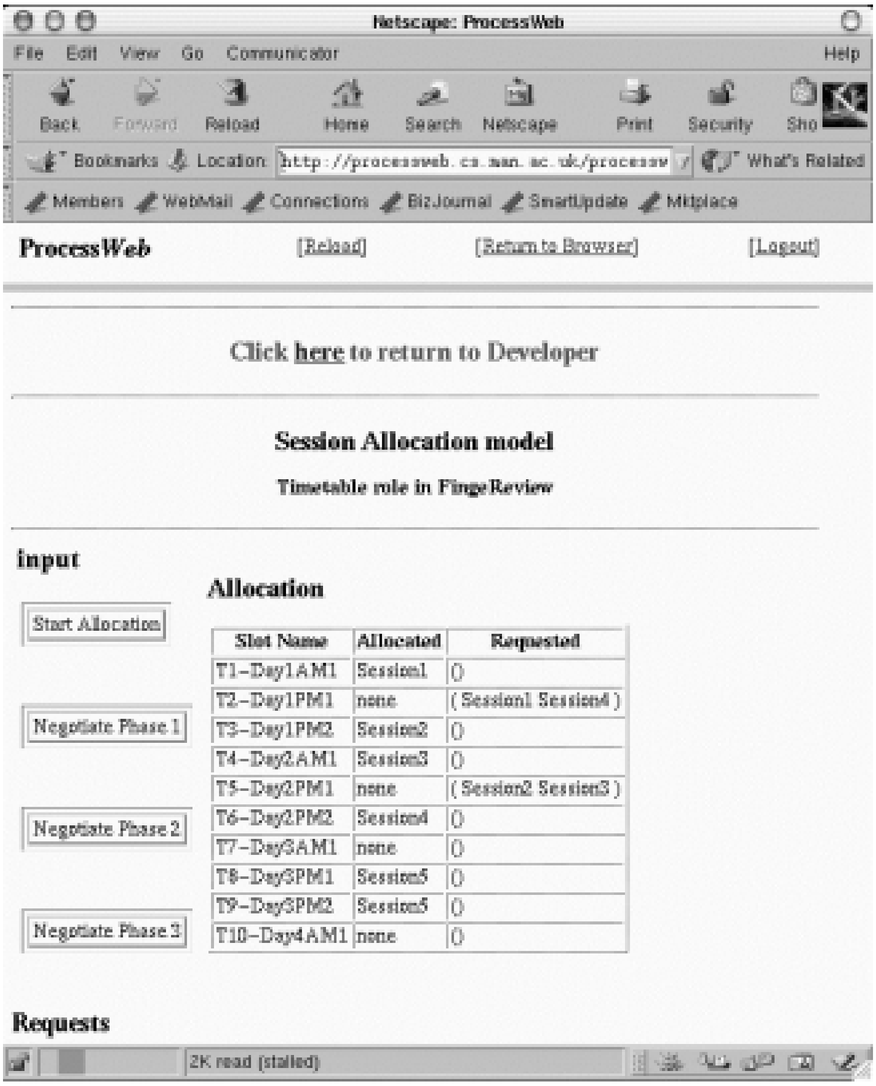
Select a slot, and then press button below

request relinquish

Allocation

Slot Name	Allocated	Requested
T1-Day1AM1	none	(Session1)
T2-Day1PM1	none	(Session1 Session4)
T3-Day1PM2	none	(Session2)
T4-Day2AM1	none	(Session3)
T5-Day2PM1	none	(Session2 Session3)
T6-Day2PM2	none	(Session4)
T7-Day3AM1	none	()
T8-Day3PM1	none	(Session5)
T9-Day3PM2	none	(Session5)
T10-Day4AM1	none	()

Figure E.4: Session Chair 1 selects timeslots for session 1 in ProcessWeb


Netscape: ProcessWeb
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Sho
Bookmarks Location: http://processweb.cs.man.ac.uk/processw
What's Related
Members WebMail Connections BizJournal SmartUpdate Mktplace
ProcessWeb
[Reload]
[Return to Browser]
[Logout]
Start Allocation
Negotiate Phase 1
Negotiate Phase 2
Negotiate Phase 3
2K read (stalled)

Click here to return to Developer

Session Allocation model

Timetable role in FingeReview

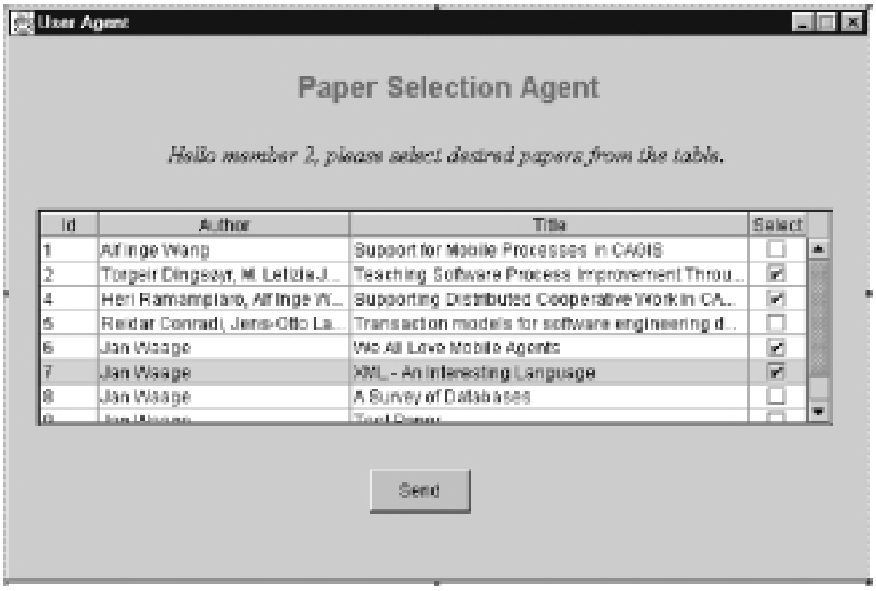
input

Allocation

Slot Name	Allocated	Requested
T1-Day1AM1	Session1	()
T2-Day1PM1	none	(Session1 Session4)
T3-Day1PM2	Session2	()
T4-Day2AM1	Session3	()
T5-Day2PM1	none	(Session2 Session3)
T6-Day2PM2	Session4	()
T7-Day3AM1	none	()
T8-Day3PM1	Session5	()
T9-Day3PM2	Session5	()
T10-Day4AM1	none	()

Requests

Figure E.5: Timetable role for allocating timeslots for sessions in ProcessWeb

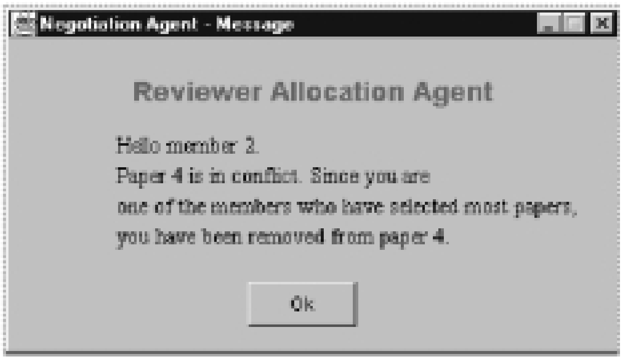


Paper Selection Agent

Hello member 2, please select desired papers from the table.

Id	Author	Title	Select
1	Alf Inge Wang	Support for Mobile Processes in CAGIS	☐
2	Torgeir Dingsøyr, M. Letizia J...	Teaching Software Process Improvement Throu...	✓
4	Heri Ramampiaro, Alf Inge W...	Supporting Distributed Cooperative Work in CA...	✓
5	Reidar Conradi, Jens-Otto La...	Transaction models for software engineering d...	☐
6	Jan Waage	We All Love Mobile Agents	✓
7	Jan Waage	XML - An Interesting Language	✓
8	Jan Waage	A Survey of Databases	☐
[illegible]	[illegible]	[illegible]	☐

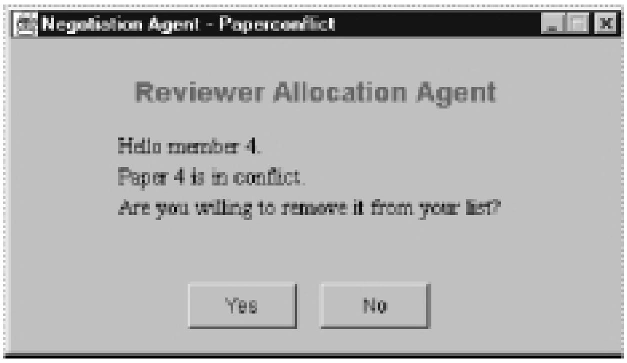
Figure E.6: Paper selection agent in CAGIS



Reviewer Allocation Agent

Hello member 2.
Paper 4 is in conflict. Since you are
one of the members who have selected most papers,
you have been removed from paper 4.

Figure E.7: User is forced to “unselect” paper in CAGIS

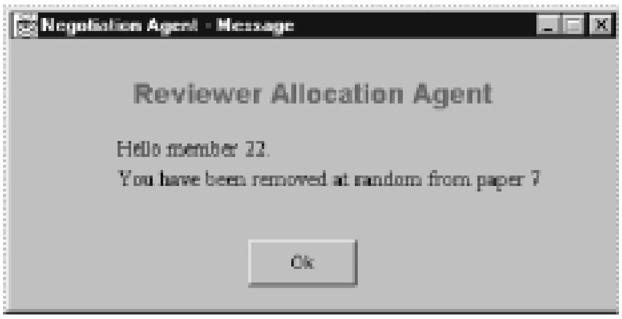


Negotiation Agent - Paperconflict

Reviewer Allocation Agent

Hello member 4.
Paper 4 is in conflict.
Are you willing to remove it from your list?

Figure E.8: User asked to “unselect” paper in CAGIS



Negotiation Agent - Message

Reviewer Allocation Agent

Hello member 22.
You have been removed at random from paper 7

Figure E.9: A random user is selected to “unselect” paper in CAGIS

APPENDIX F

Endeavors Process Model for the Conference Scenario

F.1 Graphical Process Models for the Scenario

In Endeavors, the activity network is modelled in the Endeavors Network Artist. This section shows screenshots from the process model for the conference scenario. In figure F.1, the high-level process is shown, where the rests of the figures shows decompositions of this process.

F.2 Activity Handlers for the Scenario

There are two different activity handlers used for the conference organising process. The activity handler Browser Execute shown in figure F.11, is used to start a web-browser showing the user what to do or asking for information in a HTML-form. The second handler Allocate Execute shown in figure F.12, is used to solve paper allocation conflicts for reviewing. Similar handlers are also used to deal with other allocation problems.

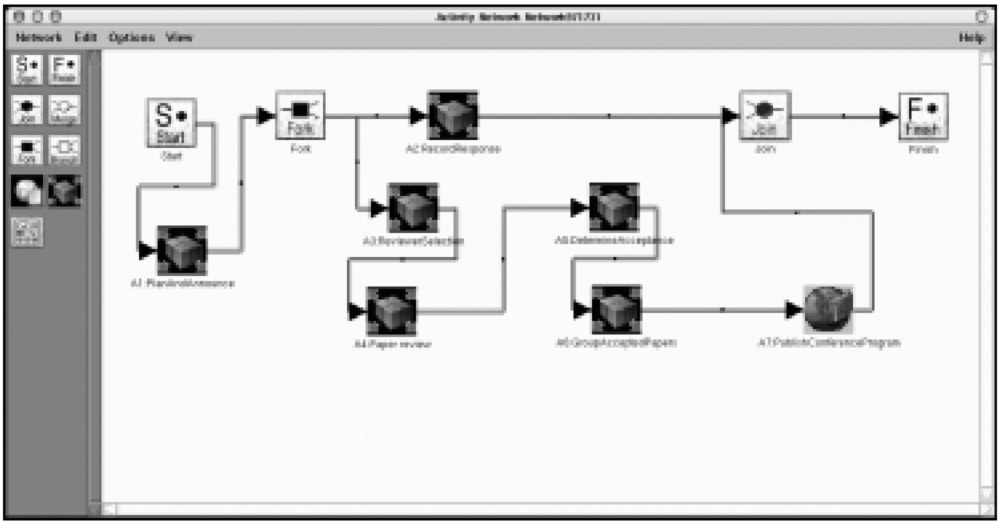


Figure F.1: High-level process model of the conference process in Endeavors

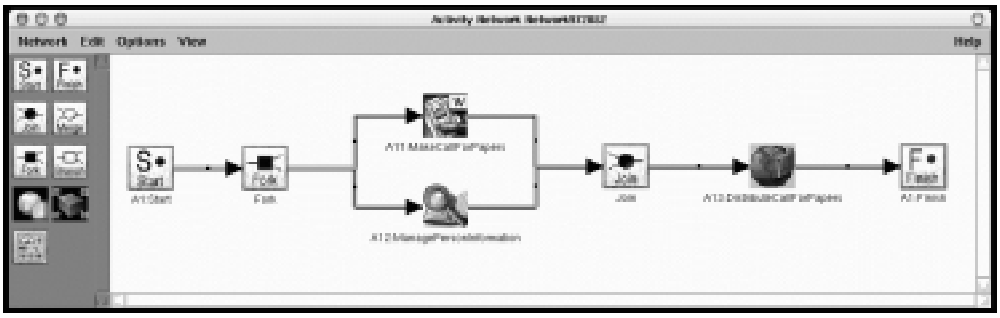


Figure F.2: Activity A1:Plan and announce conference decomposed

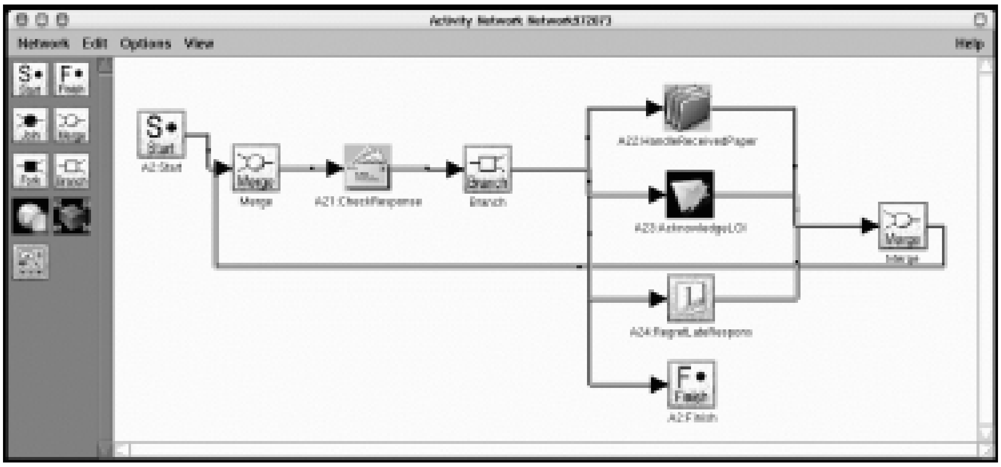


Figure F.3: Activity A2:Record response decomposed

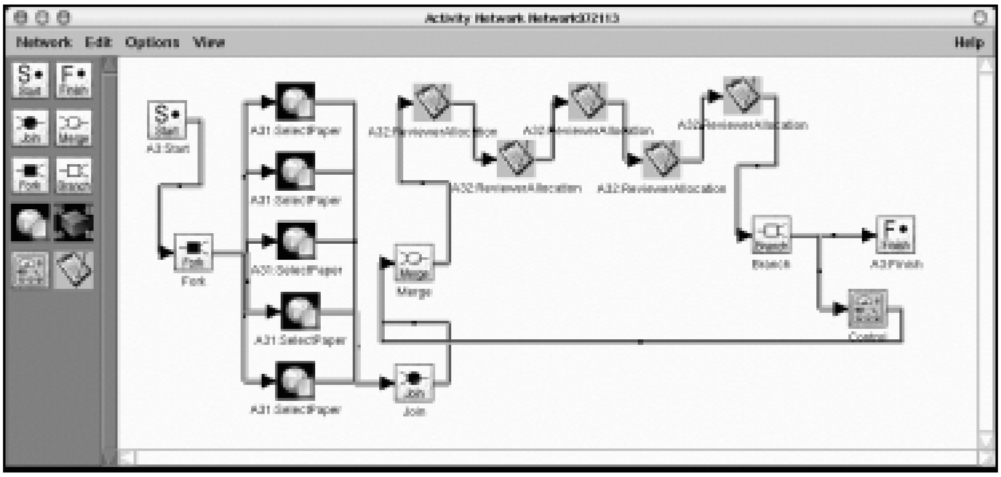


Figure F.4: Activity A3:Reviewer selection decomposed

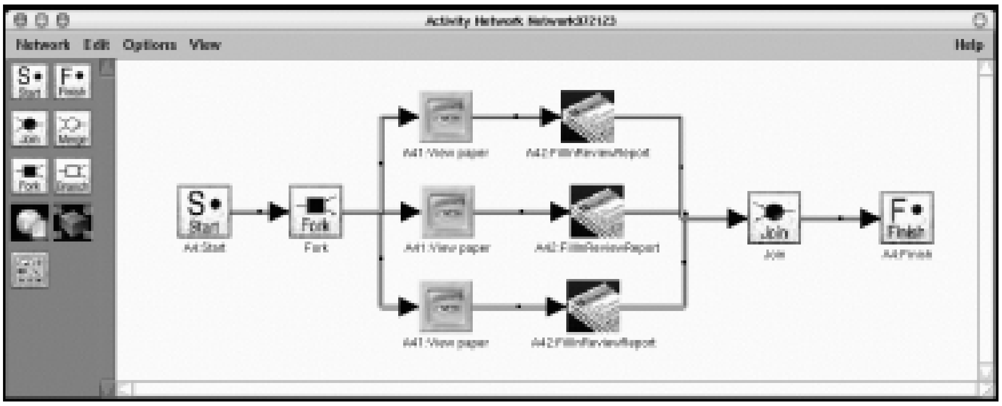


Figure F.5: Activity A4:Paper review decomposed

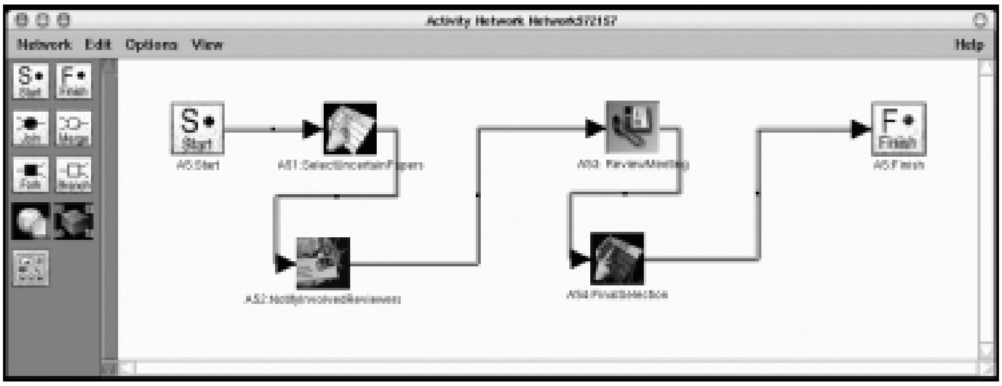


Figure F.6: Activity A5:Determine acceptance of papers decomposed

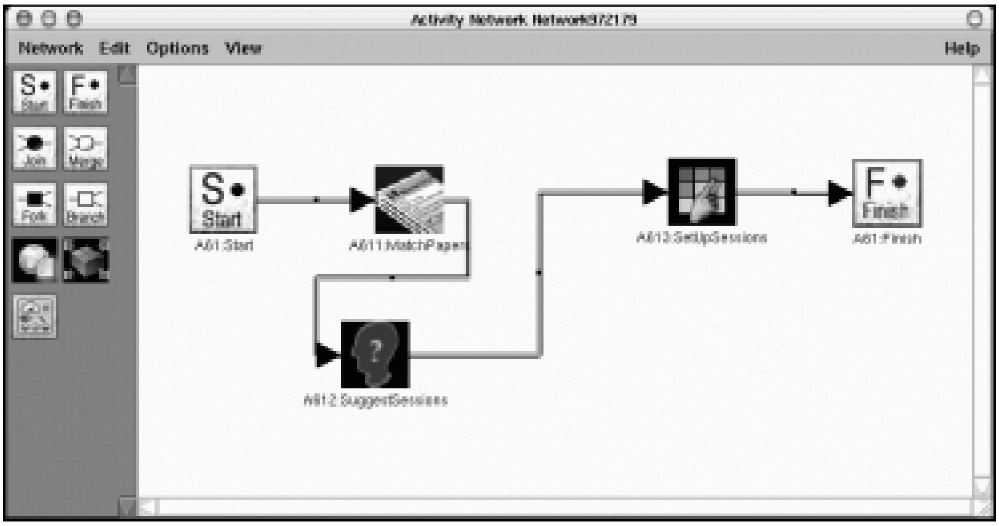


Figure F.7: Activity A6.1:Suggest sessions decomposed

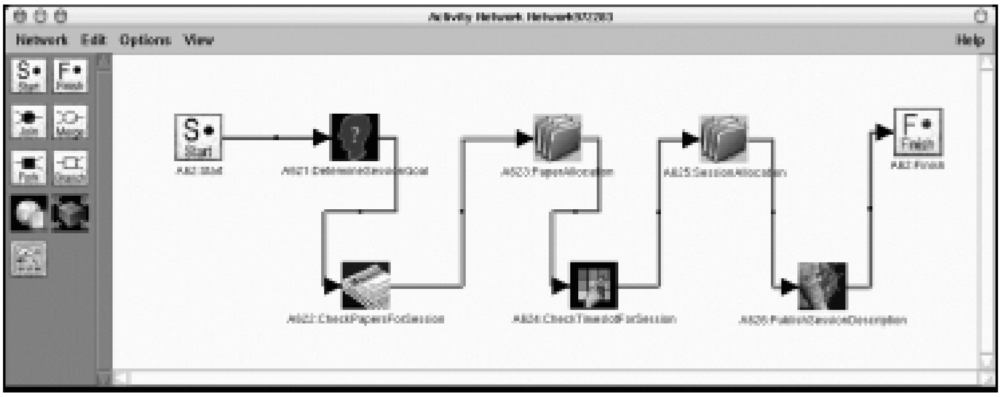

Network
Edit
Options
View
Help
Start
Finish
A6.2:Start
A6.21:DetermineSessionGoal
A6.23:PapersAllocation
A6.25:SessionAllocation
A6.2:Finish
A6.22:CheckPapersForSession
A6.24:CheckTimeslotForSession
A6.26:PublishSessionDescription

Figure F.8: Activity A6.2:Select papers and Plan sessions decomposed

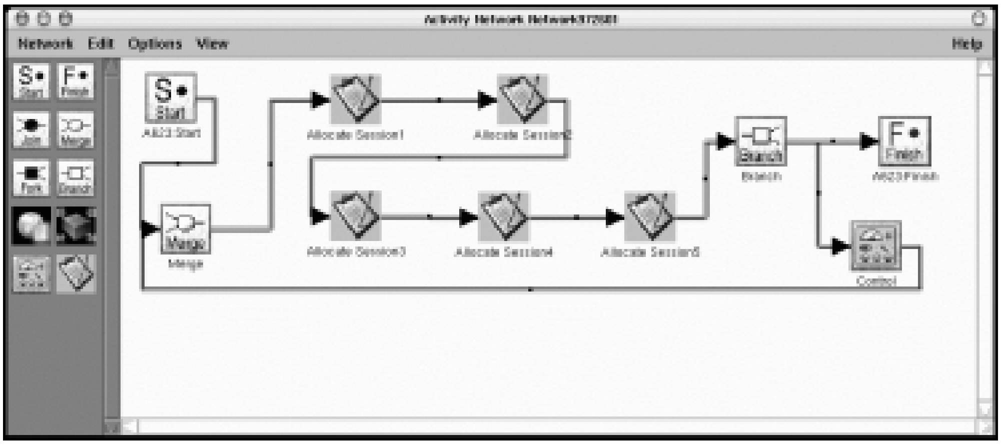


Figure F.9: Activity A6.2.3:Paper allocation decomposed

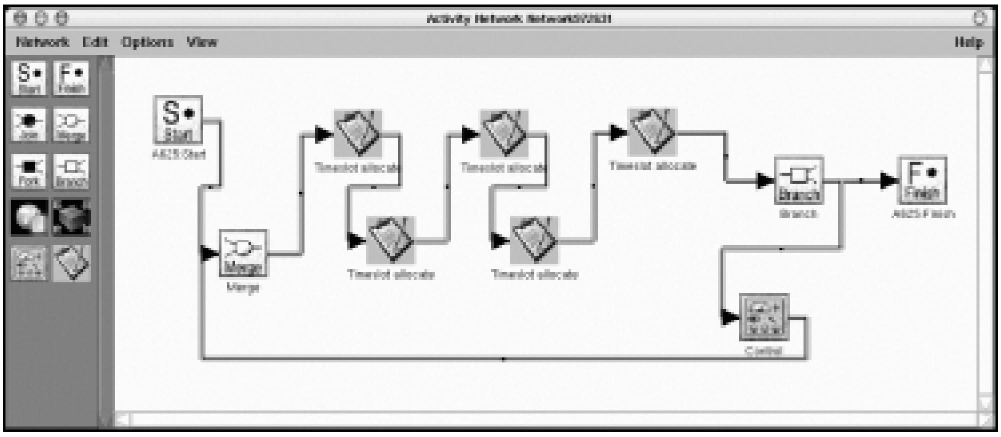


Figure F.10: Activity A6.2.5:Session allocation decomposed

```
import BrowserControl;
import WaitDialog;

import java.awt.*;
import java.applet.Applet;
import java.util.Hashtable;
import java.util.Vector;
import Endeavors.Foundation.Services.CategoryHandler;
import Endeavors.Foundation.EndFoundation;


public class Execute extends CategoryHandler
{

  public void run()
  {
     Integer id = (Integer) args.get("NodeID");

     String title = EndFoundation.GetName(id);
     String url = EndFoundation.Retrieve(id,"URI");
     String role = EndFoundation.Retrieve(id,"AssignedTo");
     String description = EndFoundation.Retrieve(id,"Description");

     BrowserControl.displayURL(url);

     String dialogstring = "Description: " + description + "\n"+
         "Action: Execute tasks in the web-browser and press Done.";

     WaitDialog wd = new WaitDialog(role,dialogstring,"Done");
     wd.waitForDone();

  }
} // Execute
```

Figure F.11: Activity handler for initiating a web-page

```
import SelectDialog;
import PaperDatabase;

import java.io.*;
import java.awt.*;
import java.applet.Applet;
import java.util.*;
import Endeavors.Foundation.Services.CategoryHandler;
import Endeavors.Foundation.EndFoundation;


public class Execute extends CategoryHandler
{

  public void run()
  {
    Integer id = (Integer) args.get("NodeID");
    String title = EndFoundation.GetName(id);
    String url = EndFoundation.Retrieve(id,"URI");
    String role = EndFoundation.Retrieve(id,"AssignedTo");
    String description = EndFoundation.Retrieve(id,"Description");
    String automate = EndFoundation.Retrieve(id,"Automate");

    PaperDatabase paperdatabase = new PaperDatabase();
    boolean conflict = paperdatabase.CheckConflict(role);

    if (conflict) {
      boolean most_papers = paperdatabase.CheckMostPapers(role);

      if (most_papers) {
        String dialogstring = description + "\n" + "This item will be unselected";
        SelectDialog sd = new SelectDialog(role,dialogstring,"OK");
        sd.waitForDone();
      } else {
        String dialogstring = description +"\n"+ "Do you want to unselect this item?";
        SelectDialog sd = new SelectDialog(role,dialogstring,"Unselect",url);
        sd.waitForDone();
      }
    } else {
      SelectDialog sd = new SelectDialog(role,"No allocation conflicts","OK");
      sd.waitForDone();
    }

  }
} // Execute
```

Figure F.12: Activity handler for dealing with paper allocation conflicts

APPENDIX G

ProcessWeb Process Model for the Conference Scenario

This section is divided into two parts. The first section contains the PML code for the individual activities modelled in ProcessWeb. The second section contains the PML code for the cooperative activities modelled in ProcessWeb.

G.1 Process Model for Individual Activities

```
! Review model
!
! This is a simple process model of a paper review process.
! The process has four roles that interact with each other.



! SessionChair Role
! =================

SessionChair isa HKClient2 with
resources
        giveNotification : giveport tableof Any   ! Interaction with PCChair (output)
        takeNotification : takeport tableof Any   ! Interaction with PCChair (input)

        state: Int := 0
        result: Int := 0

        wwwFile : String
        receiveTable: tableof Any:= tableof Any ()
        sendTable : tableof Any  := tableof Any ()
        parseTable : tableof Any := tableof Any ()

actions
        ! Set initial values for some variables
        init : {
                wwwFile := configuration.processWebHome ++ configuration.templateDirectory ++
                           'conference/sessionchair.html';
                parseTable('$modelName')    := modelName;
                parseTable('$roleName')     := roleName;
                state := 1;
        } when state = 0 & configuration ~=nil
        initial_state: {
                state := 2;
                parseTable('$message') := '';
                parseTable('$state') := state;
                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);
        } when state = 1

        receive_selection: {
                Take(
                        gram=cgi_data,
                        interaction=userRolePorts.userTakeport);

                if cgi_data ~= nil & cgi_data('a621') ~= nil then
                        state := 4;
                end if;

                if cgi_data ~= nil & cgi_data('a622') ~= nil then
                        state := 5;
                end if;

                if cgi_data ~= nil & cgi_data('a623') ~= nil then
                        state := 6;
                end if;

                if cgi_data ~= nil & cgi_data('a624') ~= nil then
                        state := 7;
                end if;

                if cgi_data ~= nil & cgi_data('a625') ~= nil then
                        state := 8;
                end if;

                if cgi_data ~= nil & cgi_data('a626') ~= nil then
                        state := 9;
                        sendTable('state') := 'publish session';

                        ! Send notification to PCChair
                        GiveCopy(
                                gram=sendTable,
                                interaction=giveNotification);
                end if;

                parseTable('$state') := state;
                SendToUser(
                        gram=WWW_file(
                        file_name=wwwFile,
                        replacement_table=parseTable),
                        connection=userRolePorts);

                cgi_data := nil;

        } when cgi_data = nil & userRolePorts ~= nil & userRolePorts.userTakeport ~= nil

        receive_pcchair: {
                Take(
                        interaction=takeNotification,
                        gram=receiveTable);
                state := 3;
                parseTable('$state') := state;

                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);

        } when takeNotification ~= nil & state = 2
end with



! PCchair Role
! ============
PCChair isa HKClient2 with
resources
        giveMessage : giveport tableof Any ! Interaction with SessionChair role
        takeMessage : takeport tableof Any ! Interaction with SessionChair role
        giveOrders  : giveport tableof Any ! Interaction with PCMember role
        takeToken  : takeport tableof Any  ! Interaction with PCMember role

        state : Int := 0
        state2: Int := 0
        state3: Int := 0
        reviewresult : Int := 0
        a11: String
        a12: String
        paper: String
        wwwFile : String
        sendTable : tableof Any := tableof Any ()
        receiveTable : tableof Any := tableof Any ()
        parseTable : tableof Any := tableof Any ()
actions
        init: {
                wwwFile :=  configuration.processWebHome ++ configuration.templateDirectory ++
                            'conference/pcchair.html';
                parseTable('$modelName') := modelName;
                parseTable('$roleName') := roleName;
                parseTable('$message') :='';
                state := 1;
        } when  state=0 & configuration ~= nil


        waiting: {
                ! A11 & A12
                state := 2;
                parseTable('$state') := state;
                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);

        } when state = 1

        receive_selection: {
                Take(
                        gram=cgi_data,
                        interaction=userRolePorts.userTakeport);
                if cgi_data('a11') ~= nil | cgi_data('a12') ~= nil then
                        if cgi_data ~= nil & cgi_data('a11') ~= nil then
                                a11 := 'checked';
                        end if;
                        if cgi_data ~= nil & cgi_data('a12') ~= nil then
                                a12 := 'checked';
                        end if;
                        if a11 = 'checked' & a12 = 'checked' then
                                ! A13
                                state := 3;
                        end if;
                end if;
                if cgi_data ~= nil & cgi_data('a13') ~= nil then
                        ! A21 Check Response
                        state := 4;
                        sendTable('state') := 'review';
                        GiveCopy(
                                gram=sendTable,  ! Give orders to PCMember
                                interaction=giveOrders);
                end if;
                if cgi_data ~= nil & cgi_data('Response') ~= nil then
                        if cgi_data('Response') = 'Paper received' then
                                ! A22 Handle received paper
                                state := 5;
                        end if;
                        if cgi_data('Response') = 'Letter of intent' then
                                ! A23 Acknowledge letter of intent
                                state := 6;
                        end if;
                        if cgi_data('Response') = 'Late response' then
                                ! A24 Regret late response
                                state := 7;
                        end if;
                end if;

                if cgi_data ~= nil &
                   ( cgi_data('a22') ~= nil | cgi_data('a23') ~= nil | cgi_data('a24') ~= nil ) then

                        ! From either A22 | A23 | A24 Go back to A21
                        state := 4;
                end if;

                if cgi_data ~= nil & cgi_data('a51') = 'finished' then
                        state := 9;
                end if;

                if cgi_data ~= nil & cgi_data('a52') = 'finished' then
                        state := 4;
                        sendTable('state') := 'reviewmeeting';
                        GiveCopy(
                                gram=sendTable,  ! Give orders to PCMember
                                interaction=giveOrders);
                end if;

                if cgi_data ~= nil & cgi_data('a54') = 'finished' then
                        state := 11;
                end if;

                if cgi_data ~= nil & cgi_data('a611') = 'finished' then
                        state := 12;
                end if;
                if cgi_data ˜= nil & cgi_data('a612') = 'finished' then
                        state := 13;
                end if;

                if cgi_data ˜= nil & cgi_data('a613') = 'finished' then
                        state := 4;
                        sendTable('state') := 'session';
                        GiveCopy(
                                gram=sendTable,  ! Give orders to PCMember
                                interaction=giveMessage);
                end if;

                if cgi_data ˜= nil & cgi_data('a7') = 'finished' then
                        state := 20;
                end if;

                ! Generate UI feedback to PCChair
                parseTable('$a11')   := a11 as String;
                parseTable('$a12')   := a12 as String;
                parseTable('$state') := state;

                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);
                cgi_data := nil;

        }  when cgi_data = nil & userRolePorts ˜= nil & userRolePorts.userTakeport ˜= nil


        receive_pcmember: {
                Take(
                        gram=receiveTable,
                        interaction=takeToken);


                if receiveTable('state') = 'paper reviewed' then
                        state := 8;
                end if;
                if receiveTable('state') = 'finished reviewmeeting' then
                        state := 10;
                end if;


                ! Generate UI feedback to PCChair
                parseTable('$message') := '';
                parseTable('$state') :=state;
                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);
        } when takeToken ˜= nil & state > 2


        receive_sessionchair: {
                Take(
                        gram=receiveTable,
                        interaction=takeMessage);
                if receiveTable('state') = 'publish session' then
                        state := 14;
                end if;

                ! Generate UI feedback to PCChair
                parseTable('$message') := '';
                parseTable('$state') :=state;
                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);

        } when takeMessage ˜= nil & state > 2

end with



! PCmember Role
! =============
PCMember isa HKClient2 with
resources
        giveMessage : giveport tableof Any   ! Interaction with PCChair role
        takeOrders :  takeport tableof Any   ! Interaction with PCChair role

        state: Int := 0
        paper: Int := 0

        wwwFile : String
        receiveTable : tableof Any := tableof Any ()
        sendTable : tableof Any := tableof Any ()
        parseTable : tableof Any := tableof Any ()


actions
        init: {
                wwwFile :=  configuration.processWebHome ++ configuration.templateDirectory ++
                            'conference/pcmember.html';
                ! Need to define these to avoid trouble with HTML output
                parseTable('$modelName')    := modelName;
                parseTable('$roleName')     := roleName;
                parseTable('$message')      := '';

                state := 1;
        } when  state=0

        waiting: {
                state := 2;
                parseTable('$state') := state;
                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);
        } when state = 1

        receive_pcchair: {
                Take(
                        gram=receiveTable,
                        interaction=takeOrders);
                if (receiveTable('state') = 'review') then
                        state := 3;
                end if;
                if (receiveTable('state') = 'reviewmeeting') then
                        state := 7;
                end if;

                parseTable('$state')   := state;

                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);
        } when takeOrders ~= nil

        receive_selection: { ! send_paper
                Take(
                        gram=cgi_data,
                        interaction=userRolePorts.userTakeport);
                if cgi_data ~= nil & cgi_data('a31') = 'finished' then
                        state := 4;
                end if;

                if cgi_data ~= nil & cgi_data('a32') = 'finished' then
                        state := 5;
                end if;

                if cgi_data ~= nil & cgi_data('a41') = 'finished' then
                        state := 6;
                end if;

                if cgi_data ~= nil & cgi_data('a42') = 'finished' then
                        state := 2;
                        receiveTable('state') := 'paper reviewed';
                        GiveCopy(
                                gram=receiveTable,  ! Send token back to PCChair
                                interaction=giveMessage);
                        end if;

                if cgi_data ~= nil & cgi_data('a53') = 'finished' then
                        state := 2;
                        receiveTable('state') := 'finished reviewmeeting';
                        GiveCopy(
                                gram=receiveTable,  ! Send token back to PCChair
                                interaction=giveMessage);
                        end if;


                ! Give UI feedback to the PCMember role
                parseTable('$state') := state;
                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                        connection=userRolePorts);
                cgi_data := nil;
        } when cgi_data = nil & userRolePorts ~= nil & userRolePorts.userTakeport ~= nil
end with
```

```
!
! Need a BOOT role to instanciate the different roles
! in the process model
! ===============================================
ReviewBoot isa HKClient2 with

resources
        userRolePorts_sessionchair, userRolePorts_pcchair : ModelUserRecord
        userRolePorts_pcmember : ModelUserRecord

        sessionchair, pcchair, pcmember : Role

        ! Interactions between the Roles
        sessionchairGiveNotification, pcchairGiveOrders, pcmemberGiveOrders: giveport tableof Any
        pcchairGiveMessage,pcmemberGiveMessage: giveport tableof Any

        pcchairTakeMessage, pcmemberTakeOrders : takeport tableof Any
        pcmemberTakeToken : takeport tableof Any
        sessionchairTakeNotification, pcchairTakeToken : takeport tableof Any

        sendReq_sessionchair, sendReq_pcchair, sendReq_pcmember : giveport HKRequest

        reviewprocessClasses : Classes

        startRoleBindings_sessionchair, startRoleBindings_pcchair : tableof Any
        startRoleBindings_pcmember : tableof Any

        warnings: String

actions
        start_reviewprocess: {
                ! Create the interactions between roles
                NewInteraction(
                        giver=sessionchairGiveNotification,
                        taker=pcchairTakeMessage);
                NewInteraction(
                        giver=pcchairGiveMessage,
                        taker=sessionchairTakeNotification);
                NewInteraction(
                        giver=pcchairGiveOrders,
                        taker=pcmemberTakeOrders);
                NewInteraction(
                        giver=pcmemberGiveMessage,
                        taker=pcchairTakeToken);

                ! Create duplicates of the Housekeeper Interaction <sendReq>
                Duplicate(
                        original=sendReq,
                        duplicate=sendReq_sessionchair);
                Duplicate(
                        original=sendReq,
                        duplicate=sendReq_pcchair);
                Duplicate(
                        original=sendReq,
                        duplicate=sendReq_pcmember);

                ! Create User Role for each role defined
                CreateUserRole(
                        nodeID=nodeID,
                        roleName='SessionChair',
                        managerGiveport=managerGiveport,
                        newUser=userRolePorts_sessionchair);
                CreateUserRole(
                        nodeID=nodeID,
                        roleName='PCChair',
                        managerGiveport=managerGiveport,
                        newUser=userRolePorts_pcchair);
                CreateUserRole(
                        nodeID=nodeID,
                        roleName='PCMember',
                        managerGiveport=managerGiveport,
                        newUser=userRolePorts_pcmember);


                ! Set up the bindings for the four roles involved. The bindings
                ! are held in a table - the keys of the table should correspon
                ! to variable names used in the role classes.
                startRoleBindings_sessionchair := tableof Any (
                        'roleName' -> 'Reviewprocess SessionChair',
                        'modelName' -> modelName,
                        'giveNotification' -> sessionchairGiveNotification,
                        'takeNotification' -> sessionchairTakeNotification,
                        'sendReq' -> sendReq_sessionchair,
                        'userRolePorts' -> userRolePorts_sessionchair);
                startRoleBindings_pcchair := tableof Any (
                        'roleName' -> 'Reviewprocess PCChair',
                        'modelName' -> modelName,
                        'giveMessage' -> pcchairGiveMessage,
                        'giveOrders' -> pcchairGiveOrders,
                        'takeMessage' -> pcchairTakeMessage,
                        'takeToken' -> pcchairTakeToken,
                        'sendReq' -> sendReq_pcchair,
                        'userRolePorts' -> userRolePorts_pcchair);
                startRoleBindings_pcmember := tableof Any (
                        'roleName' -> 'Reviewprocess PCMember',
                        'modelName' -> modelName,
                        'giveMessage' -> pcmemberGiveMessage,
                        'giveOrders' -> pcmemberGiveOrders,
                        'takeOrders' -> pcmemberTakeOrders,
                        'takeToken' -> pcmemberTakeToken,
                        'sendReq' -> sendReq_pcmember,
                        'userRolePorts' -> userRolePorts_pcmember);


                ! Get the set of compiled classes from the environment
                ! of this role. This is necessary for the StartRole Action
                GetRoleClasses(outputClasses=reviewprocessClasses);


                ! Create the role instances. The binding tables should
                ! ensure that all Interactions and User Roles are
                ! bound to the correct resources.
                StartRole(
                        className='SessionChair',
                        roleInst=sessionchair,
                        inputClasses=reviewprocessClasses,
                        bindings=startRoleBindings_sessionchair,
                        warnings=warnings);
                StartRole(
                        className='PCChair',
                        roleInst=pcchair,
                        inputClasses=reviewprocessClasses,
                        bindings=startRoleBindings_pcchair,
                        warnings=warnings);
                StartRole(
                        className='PCMember',
                        roleInst=pcmember,
                        inputClasses=reviewprocessClasses,
                        bindings=startRoleBindings_pcmember,
                        warnings=warnings);


                ! Let Housekeeper know about the new Roles.
                GiveCopy(
                        interaction=sendReq,
                                gram=HKRequest(
                                        roleName='SessionChair',
                                        roleAssoc=sessionchair,
                                        request='ADD'));
                GiveCopy(
                        interaction=sendReq,
                                gram=HKRequest(
                                        roleName='PCChair',
                                        roleAssoc=pcchair,
                                        request='ADD'));
                GiveCopy(
                        interaction=sendReq,
                                gram=HKRequest(
                                        roleName='PCMember',
                                        roleAssoc=pcmember,
                                        request='ADD'));

        } when start_reviewprocess = nil
end with
```

G.2 Cooperative Activities Modelled in ProcessWeb

The cooperative activities described in the conference scenario are basically the same, because they use the same negotiation strategy for solving conflicts. We will therefor show the PML code just for two cooperative activities in this section: **A6.2.4:Check timeslot for session** and **A6.2.5:Session allocation**. These two activities was modelled by Mark R. Greenwood, Informatics Process Group at the University of Manchester.

The process modelling code for the activities A6.2.4 and A6.2.5 are described in the file *afsession.pml*. Note that the file *devstart.pml* is needed in order to compile this file, but this file was only used for debugging purposes.

G.2.1 afsession.pml

```
! Session allocation - afsession.pml
! RMG 16-10-00

! requires HKClient3, StartHK3 (devstart.pml)

RequestSlotMsg isa Entity with
parts
  sessionName : String
  slotName : String
  remove : Bool := false
end with

SlotInfo isa Entity with
parts
  allocated : String
  possible : collof String
end with

TimeTableMsg isa Entity with
parts
  ! allocation : tableof SlotInfo
  alloc_html : String
  message : String
end with

AllocHTML isa Action with
out
  alloc : tableof SlotInfo
  alloc_html : String
  slotnames : collof String

resources
  slot, session : String

parts

do_it: {
        alloc_html := '<table border>\n<tr><th>Slot Name</th> <th>Allocated</th> <th>Requested</th>\n';
        forevery slot in slotnames do
                alloc_html := alloc_html ++ '<tr><td>' ++ slot ++ '</td>';
                if alloc(slot).allocated = nil then
                        alloc_html := alloc_html ++ '<td> none </td>';
                else
                        alloc_html := alloc_html ++ '<td>' ++ alloc(slot).allocated ++ '</td>';
                end if;
                if lengthof alloc(slot).possible = 0 then
                        alloc_html := alloc_html ++ '<td>()</td></tr>\n';
                elsif lengthof alloc(slot).possible = 1 then
                        alloc_html := alloc_html ++ '<td>( ' ++ alloc(slot).possible(1) ++ ' )</td></tr>\n';
                else
                        alloc_html := alloc_html ++ '<td>( ';
                        forevery session in alloc(slot).possible do
                                alloc_html := alloc_html ++ session ++ ' ';
                        end do;
                        alloc_html := alloc_html ++ ')</td></tr>\n';
                end if
        end do;
        alloc_html := alloc_html ++ '</table>\n';

} when do_it = nil

end with

FindMost isa Action with
out
  conflict : SlotInfo
  requests : tableof collof String
  res : String

resources
  max : Int := 0
  sess : String

parts

find:{
        forevery sess in conflict.possible do
                if lengthof( requests(sess) ) > max then
                        max := lengthof( requests(sess) );
                        res := sess;
                elsif lengthof( requests(sess) ) = max then
```

```
                        res := 'multiple';
                end if;
        end do;
}
when max = 0

preconds
  conflict ~= nil & requests ~= nil

postconds
  res = 'multiple' | res memberof conflict.possible

end with


RemoveRequests isa Action with
in
  sess : String
out
  alloc : tableof SlotInfo

resources
  slot : String
  slotnames : collof String
  index : Int

parts
remove:{
        Domain( table = alloc, collection = slotnames );
        forevery slot in slotnames do
                if sess memberof alloc(slot).possible then
                        RemoveAtIndex(
                                collection = alloc(slot).possible,
                                index = sess indexin alloc(slot).possible);
                end if;
        end do;
}
when remove = nil

end with


TimeTable isa HKClient3 with
resources
  slots : collof String         ! initialised at startup
  sessions : collof String      ! initialised at startup

  ! require lengthof(slots) = lengthof(sessions) * max_sessions
  max_sessions : Int := 2

  allocation : tableof SlotInfo := tableof SlotInfo ()
  ! the domain of allocation = slots

  ! requests and allocated could be derived from allocation
  ! but it is easier to have them separate to retrieve information
  ! by session name
  requests  : tableof collof String := tableof collof String ()
  allocated : tableof collof String := tableof collof String ()

  requestTP : takeport RequestSlotMsg
  updateGPs : tableof giveport TimeTableMsg

  phase : String := 'mark'
  allocate_pending, finished : Bool := false

  request : RequestSlotMsg
  reply   : TimeTableMsg
  slot    : String
  session : String
  index   : Int

  allocation_html : String

actions

init: {
  if wwwFile = nil then
    wwwFile := configuration.processWebHome ++ configuration.templateDirectory ++  'examples/timet.htm';
  end if;
  parseTable('$lastmsg') := 'No message has been sent';
  parseTable('$modelName') := modelName;  ! <modelName> should have been set by Developer
  forevery slot in slots do
        allocation(slot) := SlotInfo(
                                allocated = nil,
                                possible = collof String () );
  end do;
  forevery session in sessions do
        requests(session) := collof String();
        allocated(session) := collof String();
  end do;
  AllocHTML( alloc=allocation, alloc_html=allocation_html,
                slotnames=slots );
  parseTable('$allocation') := allocation_html;
  SendToUser(
      gram=WWW_file(
               file_name=wwwFile,
               replacement_table=parseTable),
      connection=userRolePorts);
} when
    init = nil &           ! i.e. only do the <init> action once
    configuration ~= nil   ! <configuration> is set by an action in HKClient3

mark_timeslots: {
        Take( interaction = requestTP, gram = request );
        parseTable('$lastmsg') := request;
        ! assume request is fully-formed, no nil parts
        ! and request.slotName isin slots
        if request.remove | request.slotName memberof requests(request.sessionName) then
                GiveCopy( interaction = updateGPs(request.sessionName),
                          gram = TimeTableMsg(
                                        message = request.slotName ++ ' request/relinquish ignored')
                        );
        else
                AddToCollection(
                        collection = allocation(request.slotName).possible,
                        item = request.sessionName );
                AddToCollection(
                        collection = requests(request.sessionName),
                        item = request.slotName );
                AllocHTML( alloc=allocation, alloc_html=allocation_html,
                                slotnames=slots );
                parseTable('$allocation') := allocation_html;
                forevery session in sessions do
                        reply := TimeTableMsg(alloc_html=allocation_html);
                        if session = request.sessionName then
                                reply.message := request.slotName ++ ' request accepted';
                        else
                                reply.message := 'update';
                        end if;
                        Give( interaction = updateGPs(session),
                                gram = reply );
                end do;
        end if;
        ! parseTable('$allocation') := allocation;
        parseTable('$requests') := requests;
        SendToUser(
                gram=WWW_file(
                        file_name=wwwFile,
                        replacement_table=parseTable),
        connection=userRolePorts);
}
when phase = 'mark'

receive_selection:{
        Take(gram=cgi_data, interaction=userRolePorts.userTakeport);
        if cgi_data = nil then
                cgi_data := tableof String(); ! for ease of parsing
        end if;

        phase := cgi_data('phase');
}
when userRolePorts ~= nil

! First deal with allocation where there are no conflicts
! note one alloaction may lead to others as other requests are removed
! this action part also deals with allocation during conflict

first_alloc: {
        index := 1;
        while index <= lengthof(slots) do
                slot := slots(index);
                if lengthof( allocation(slot).possible ) = 1 then
                        session := allocation(slot).possible(1);
                        if lengthof( allocated(session) ) < max_sessions then
                                allocation(slot).allocated := session;
                                AddToCollection(
                                        collection = allocated(session),
                                        item = slot );
                                allocation(slot).possible := collof String();
                                RemoveAtIndex(
                                        collection = requests( session ),
                                        index = slot indexin requests(session) );
                                reply := TimeTableMsg(
                                        message = slot ++ ' allocated' );
                                Give( interaction = updateGPs(session),
                                     gram = reply );
                        end if;
                        if lengthof( allocated(session) ) = max_sessions then
                                requests(session) := collof String();
                                RemoveRequests(
                                        sess = session,
                                        alloc = allocation );
                                index := 1
                        else
```

```
                                index := index + 1
                        end if;
                else
                        index := index + 1
                end if;

        end do;
        AllocHTML( alloc=allocation, alloc_html=allocation_html,
                        slotnames = slots );
        parseTable('$allocation') := allocation_html;
        forevery session in sessions do
                reply := TimeTableMsg(
                                alloc_html=allocation_html,
                                message = 'alloc update');
                Give( interaction = updateGPs(session),
                        gram = reply );
        end do;
        ! parseTable('$allocation') := allocation;
        parseTable('$requests') := requests;
        SendToUser(
                gram=WWW_file(
                        file_name=wwwFile,
                        replacement_table=parseTable),
        connection=userRolePorts);
        allocate_pending := false;
        if phase = 'no_conflict' then
                phase := 'negotiate';
        end if;
}
when phase = 'no_conflict' | allocate_pending

neg1:{
        forevery slot in slots do
                if lengthof(allocation(slot).possible) > 1 then
                        FindMost( conflict = allocation(slot),
                                  requests = requests,
                                  res = session );
                        if session ~= 'multiple' then
                                RemoveAtIndex(
                                        collection = allocation(slot).possible,
                                        index = session indexin  allocation(slot).possible );
                                RemoveAtIndex(
                                        collection = requests(session),
                                        index = slot indexin requests(session) );
                                GiveCopy( interaction = updateGPs(session),
                                        gram = TimeTableMsg(
                                                message = slot ++ ' relinquished negotiation phase 1')
                        );
                        end if;
                        if lengthof(allocation(slot).possible) = 1 then
                                allocate_pending := true;
                        end if;
                end if;
        end do;
        if ~ allocate_pending then
                AllocHTML( alloc=allocation, alloc_html=allocation_html,
                                slotnames = slots );
                parseTable('$allocation') := allocation_html;
                forevery session in sessions do
                        reply := TimeTableMsg(
                                        alloc_html=allocation_html,
                                        message = 'alloc update');
                        Give( interaction = updateGPs(session),
                                gram = reply );
                end do;

                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                replacement_table=parseTable),
                connection=userRolePorts);
        end if;

! move to phase neg2 supplied from user interface
        phase := 'negotiate';
}
when phase = 'neg1'

neg2:{
        forevery slot in slots do
                if lengthof(allocation(slot).possible) > 1 then
                        forevery session in allocation(slot).possible do
                                GiveCopy( interaction = updateGPs(session),
                                          gram = TimeTableMsg(
                                                        message = 'conflict ' ++ slot ++ ' please relinquish')
                                        );
                        end do;
                end if;
        end do;

        parseTable('$lastmsg') := 'Negotiation phase 2 - requests sent';
        SendToUser(
                gram=WWW_file(
                        file_name=wwwFile,
                        replacement_table=parseTable),
        connection=userRolePorts);

! alter phase to avoid looping
        phase := 'neg2b'
}
when phase = 'neg2'

unmark_timeslots: {
        Take( interaction = requestTP, gram = request );
        parseTable('$lastmsg') := request;
        ! assume request is fully-formed, no nil parts
        ! and request.slotName isin slots
        if request.remove & request.slotName memberof requests(request.sessionName) then
                slot := request.slotName;
                session := request.sessionName;
                RemoveAtIndex(
                        collection = allocation(slot).possible,
                        index = session indexin allocation(slot).possible );
                RemoveAtIndex(
                        collection = requests(session),
                        index = slot indexin requests(session) );
                GiveCopy(
                        interaction = updateGPs(session),
                        gram = TimeTableMsg(
                                message = slot ++ ' relinquish accepted')
                        );
                if lengthof( allocation(slot).possible ) = 1 then
                        allocate_pending := true;
                        ! first_alloc will run and send revised allocation
                else
                        AllocHTML( alloc=allocation, alloc_html=allocation_html,
                                        slotnames = slots );
                        parseTable('$allocation') := allocation_html;
                        forevery session in sessions do
                                reply := TimeTableMsg(
                                                alloc_html=allocation_html,
                                                message = 'alloc update');
                                Give( interaction = updateGPs(session),
                                        gram = reply );
                        end do;
                end if;

        else
                GiveCopy( interaction = updateGPs(request.sessionName),
                          gram = TimeTableMsg(
                                        message = request.slotName ++ ' relinquish/request ignored')
                        );

        end if;
        ! parseTable('$allocation') := allocation;
        parseTable('$requests') := requests;
        SendToUser(
                gram=WWW_file(
                        file_name=wwwFile,
                        replacement_table=parseTable),
        connection=userRolePorts);
}
when phase = 'neg2b'
! could have guard phase = 'neg2b' & ~allocate_pending

neg3: {
        finished := true;
        forevery slot in slots do
                if lengthof (allocation(slot).possible) > 1 then
                        session := allocation(slot).possible(1);
                        ! not very random ?
                        RemoveAtIndex(
                                collection = allocation(slot).possible,
                                index = 1 );
                        RemoveAtIndex(
                                collection = requests(session),
                                index = slot indexin requests(session) );
                        GiveCopy(
                                interaction = updateGPs(session),
                                gram = TimeTableMsg(
                                        message = slot ++ ' relinquish negotiation phase 3')
                                );
                end if;
                if lengthof( allocation(slot).possible ) = 1 then
                        allocate_pending := true;
                        finished := false;
                end if;
        end do;
        if finished then
                ! parseTable('$allocation') := allocation;
                parseTable('$requests') := requests;
                SendToUser(
                        gram=WWW_file(
                                file_name=wwwFile,
                                        replacement_table=parseTable),
                                connection=userRolePorts);
        end if;
}
when phase = 'neg3' & ~allocate_pending & ~finished

end with


SessionChair isa HKClient3 with
resources
  sessionName : String
  slots : collof String
  ! allocation : tableof SlotInfo := tableof SlotInfo ()
  alloc_html : String := 'No sessions have been requested'

  requestGP : giveport RequestSlotMsg
  updateTP : takeport TimeTableMsg

  update : TimeTableMsg
  message : RequestSlotMsg
  log : collof String := collof String()

actions

init: {
  if wwwFile = nil then
    wwwFile := configuration.processWebHome ++ configuration.templateDirectory ++  'examples/sessc.htm';
  end if;
  parseTable('$sessionName') := sessionName;
  parseTable('$msgsent') := 'No message has been sent';
  parseTable('$modelName') := modelName;  ! <modelName> should have been set by Developer

  parseTable('$allocation') := alloc_html;

  parseTable('$slots') := HTMLOptionList(
                                name = 'slot_name',
                                options = slots,
                                size = 10);

  parseTable('$log') := '';
  parseTable('$cgierr') := '';
  SendToUser(
      gram=WWW_file(
               file_name=wwwFile,
               replacement_table=parseTable),
      connection=userRolePorts);
} when
    init = nil &           ! i.e. only do the <init> action once
    configuration ~= nil   ! <configuration> is set by an action in HKClient3

rec_update: {
        Take(interaction = updateTP, gram = update );
        if update.message ~= 'update' then
                AddToCollection(
                        collection = log,
                        item = update.message );
        end if;
        if update.alloc_html ~= nil then
                alloc_html := update.alloc_html;
        end if;
        parseTable('$allocation') := alloc_html;
        parseTable('$log') := log;
        SendToUser(
                gram=WWW_file(
                        file_name=wwwFile,
                        replacement_table=parseTable),
                connection=userRolePorts);
}
when updateTP ~= nil

receive_selection:{
   Take(gram=cgi_data, interaction=userRolePorts.userTakeport);
   if cgi_data = nil then
      cgi_data := tableof String(); ! for ease of parsing
   end if;

   if cgi_data('session_cmd') = 'request' & cgi_data('slot_name') memberof slots then
        message := RequestSlotMsg(
                        sessionName = sessionName,
                        slotName = cgi_data('slot_name'));
        GiveCopy(
                interaction = requestGP,
                gram =  message);
        parseTable('$lastmsg') := message;
        parseTable('$cgierr') := '';
   elsif cgi_data('session_cmd') = 'relinquish' & cgi_data('slot_name') memberof slots then
        message := RequestSlotMsg(
                        sessionName = sessionName,
                        slotName = cgi_data('slot_name'),
                        remove = true );
        GiveCopy(
                interaction = requestGP,
                gram = message);
        parseTable('$lastmsg') := message;
        parseTable('$cgierr') := '';
   else
        parseTable('$cgierr') := cgi_data;
        SendToUser(
                gram=WWW_file(
                        file_name=wwwFile,
                        replacement_table=parseTable),
                connection=userRolePorts);
   end if;

}
when userRolePorts ~= nil


end with



ConfSetup isa StartHK3 with
resources
  sessions : collof String := collof String ('Session1','Session2','Session3','Session4', 'Session5')
  slots : collof String := collof String( 'T1-Day1AM1', 'T2-Day1PM1', 'T3-Day1PM2', 'T4-Day2AM1', 'T5-Day2PM1',
                                          'T6-Day2PM2', 'T7-Day3AM1', 'T8-Day3PM1', 'T9-Day3PM2', 'T10-Day4AM1' )

  requestSlotGP : giveport RequestSlotMsg
  requestSlotGP2 : giveport RequestSlotMsg
  requestSlotTP : takeport RequestSlotMsg

  timeGPs : tableof giveport TimeTableMsg := tableof giveport TimeTableMsg();
  timeTableGP : giveport TimeTableMsg
  timeTableTP : takeport TimeTableMsg

  sess : String

  ! standard resources startup role
  startRoleBindings : tableof Any
  rolePointer : Role

actions

start: {
        NewInteraction(giver = requestSlotGP, taker = requestSlotTP );

        forevery sess in sessions do
                Duplicate( original = requestSlotGP,
                           duplicate = requestSlotGP2 );
                NewInteraction( giver = timeTableGP,
                                taker = timeTableTP );
                timeGPs(sess) := timeTableGP;

                startRoleBindings := tableof Any (
                        'roleName' -> 'Session Chair ' ++ sess,
                        'modelName' -> modelName,
                        'slots' -> slots,
                        'sessionName' -> sess,
                        'requestGP' -> requestSlotGP2,
                        'updateTP' -> timeTableTP
                        );
                StartPWebAndUserRole(
                        roleclassName = 'SessionChair',
                        nodeID = nodeID,
                        managerGiveport = managerGiveport,
                        roleInstance = rolePointer,
                        pwebclasses = pwebclasses,
                        initbindings = startRoleBindings,
                        messages = messages,
                        sendReq = sendReq );

        end do;

        startRoleBindings := tableof Any (
                'roleName' -> 'TimeTableDB',
                'slots' -> slots,
                'sessions' -> sessions,
                'modelName' -> modelName,
                'sessionName' -> sess,
                'requestTP' -> requestSlotTP,
                'updateGPs' -> timeGPs
                );
        StartPWebAndUserRole(
                roleclassName = 'TimeTable',
                nodeID = nodeID,
                managerGiveport = managerGiveport,
                roleInstance = rolePointer,
                pwebclasses = pwebclasses,
                initbindings = startRoleBindings,
                messages = messages,
                sendReq = sendReq );
   parseTable('$messages') := messages;
   SendToUser(
        gram=WWW_file(
                file_name=wwwFile,
                replacement_table=parseTable),
        connection=userRolePorts);

}
when start = nil & init ˜= nil

end with
```

APPENDIX H

CAGIS PCE Process Model for the Conference Scenario

Here, the process model for the CAGIS PCE environment for the conference organising process will be presented. The process model consists of three main parts: the process model for the activities, the gluemodel, and the Java-code for the cooperative agents.

H.1 CAGIS Simple Process Model for the Scenario

Here is the XML-file for modelling the activities in the conference organising scenario.

```
<Process>
<Name>Conference management process</Name>
<ProcessFragment>
  <Name>A1:Plan and announce conference</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A11:MakeCallForPapers</Name>
        <Workspace>PcChair</Workspace>
        <Postlink>PcChair/A12:ManagePersonInformation</Postlink>
        <Due>None</Due>
        <Description>Create a call for papers</Description>
        <Code>http://validation.finge.com/pcchair/a11.html</Code>
  </Activity>
  <Activity>
        <Name>A12:ManagePersonInformation</Name>
        <Workspace>PcChair</Workspace>
        <Postlink>PcChair/A13:DistributeCallForPapers</Postlink>
        <Due>None</Due>
        <Description>
              Search for persons and update the Conference-personDB
        </Description>
        <Code>http://validation.finge.com/pcchair/a12.html</Code>
  </Activity>
  <Activity>
        <Name>A13:DistributeCallForPapers</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A11:MakeCallForPapers</Prelink>
        <Prelink>PcChair/A12:ManagePersonInfromation</Prelink>
        <Postlink>PcChair/A31:CheckResponseAndMaintainPersonDB</Postlink>
        <Postlink>PcMember[1-25]/A31:SelectPaper</Postlink>
        <Due>1/10/2000</Due>
        <Description>Send call-for-papers via email to
        potential partisipants</Description>
        <Code>http://validation.finge.com/pcchair/a13.html</Code>
  </Activity>
</ProcessFragment>

<ProcessFragment>
  <Name>Record response</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A21:CheckResponseAndMaintainPersonDB</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A13:DistributeCallForPapers</Prelink>
        <Postlink>PChair/A22:HandleReceivedPaper</Postlink>
        <Postlink>PcChair/A23:AcknowledgeLetterOfIntent</Postlink>
        <Postlink>PcChair/A24:RegretLateResponse</Postlink>
        <Due>None</Due>
        <Feedback>PChair/A22:HandleReceivedPaper</Feedback>
        <Feedback>PcChair/A23:AcknowledgeLetterOfIntent</Feedback>
        <Feedback>PcChair/A24:RegretLateResponse</Feedback>
        <Description>Check conference email and act accordingly</Description>
        <Code>http://validation.finge.com/pcchair/a21.html</Code>
  </Activity>
  <Activity>
        <Name>A22:HandleReceivedPaper</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A21:CheckResponseAndMaintainPersonDB</Prelink>
        <Postlink>PcChair/A21:CheckResponseAndMaintainPersonDB</Postlink>
        <Due>None</Due>
        <Description>Register received paper and store paper
        electronically</Description>
        <Code>http://validation.finge.com/pcchair/a22.html</Code>
  </Activity>
  <Activity>
        <Name>A23:AcknowledgeLetterOfIntent</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A21:CheckResponseAndMaintainPersonDB</Prelink>
        <Postlink>PcChair/A21:CheckResponseAndMaintainPersonDB</Postlink>
        <Due>None</Due>
        <Description>Send an acknowledgement by using a standard
        template</Description>
        <Code>http://validation.finge.com/pcchair/a23.html</Code>
  </Activity>
  <Activity>
        <Name>A24:RegretLateResponse</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A21:CheckResponseAndMaintainPersonDB</Prelink>
        <Postlink>PcChair/A21:CheckResponseAndMaintainPersonDB</Postlink>
        <Due>None</Due>
        <Description>Use a template to send an email with regret
        for a late response</Description>
        <Code>http://validation.finge.com/pcchair/a24.html</Code>
  </Activity>
</ProcessFragment>

<ProcessFragment>
  <Name>ReviewerSelection</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A31:SelectPaper</Name>
        <Workspace>PcMember[1-25]</Workspace>
        <Prelink>PcChair/A13:DistributeCallForPapers</Prelink>
        <Postlink>PcMember[1-25]/A32:ReviewerAllocation</Postlink>
        <Due>10/10/2000</Due>
        <Description>Select paper you want to review</Description>
        <Code>http://validation.finge.com/glueserver.cgi?
        processfragment=PcMember[1-25]/A31:SelectPaper&
        agentclass=agent.Selection.Paper</Code>
  </Activity>
  <Activity>
        <Name>A32:ReviewerAllocation</Name>
        <Workspace>PcMember[1-25]</Workspace>
        <Prelink>PcMember[1-25]/A31:SelectPaper</Prelink>
        <Postlink>PcMember[1-25]/A41:ViewPapers</Postlink>
        <Due>11/10/2000</Due>
        <Description>Check if your selection of papers is ok</Description>
        <Code>http://validation.finge.com/glueserver.cgi?
        processfragment=PCMember[1-25/A32:ReviewerAllocation&
        agentclass=agent.Negotiation.Reviewer</Code>
  </Activity>
</ProcessFragment>

<ProcessFragment>
  <Name>PaperReview</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A41:ViewPapers</Name>
        <Workspace>PcMember[1-25]</Workspace>
        <Prelink>PcMember[1-25]/A32:ReviewerAllocation</Prelink>
        <Postlink>PcMember[1-25]/A42:FillInReviewerReport</Postlink>
        <Due>25/10/2000</Due>
        <Description>View papers to be reviewed</Description>
        <Code>http://validation.finge.com/pcmember/a41.html</Code>
  </Activity>
  <Activity>
        <Name>A42:FillInReviewerReport</Name>
        <Workspace>PcMember[1-25]</Workspace>
        <Prelink>PcMember[1-25]/A41:ViewPapers</Prelink>
        <Postlink>PcChair/A51:SelectUncertainPapers</Postlink>
        <Due>1/11/2000</Due>
        <Description>Fill in a review-report for the reviewed
        papers</Description>
        <Code>http://validation.finge.com/pcmember/a42.html</Code>
  </Activity>
</ProcessFragment>

<ProcessFragment>
  <Name>DetermineAcceptanceOfPapers</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A51:SelectUncertainPapers</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcMember[1-25]/A42:FillInReviewerReport</Prelink>
        <Postlink>PcChair/A52:NotifyInvolvedReviewers</Postlink>
        <Due>5/11/2000</Due>
        <Description>Update Paper-list according to review
        results</Description>
        <Code>http://validation.finge.com/pcchair/a51.html</Code>
  </Activity>
  <Activity>
        <Name>A52:NotifyInvolvedReviewers</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A51:SelectUncertainPapers</Prelink>
        <Postlink>PcMember[1-25]/A53:ReviewMeeting</Postlink>
        <Due>5/11/2000</Due>
        <Description>Notify the involved reviewers with
        uncertain papers</Description>
        <Code>http://validation.finge.com/pcchair/a52.html</Code>
  </Activity>
  <Activity>
        <Name>A53:ReviewMeeting</Name>
        <Workspace>PcMember[1-25]</Workspace>
        <Prelink>PcChair/A52:NotifyInvolvedReviewers</Prelink>
        <Postlink>PcChair/A54:FinalPaperSelection</Postlink>
        <Due>15/11/2000</Due>
        <Description>Hold an electronic review-meeting</Description>
        <Code>http://validation.finge.com/pcmember/a53.html</Code>
  </Activity>
  <Activity>
        <Name>A54:FinalPaperSelection</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcMember[1-25]</Prelink>
        <Postlink>PcChair/A611:MatchPapers</Postlink>
        <Due>20/11/2000</Due>
        <Description>Update Paper-list and Conference-personDB
        according to review result</Description>
        <Code>http://validation.finge.com/pcchair/a54.html</Code>
  </Activity>
</ProcessFragment>

<ProcessFragment>
  <Name>Suggest Session</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A611:MatchPapers</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A54:FinalPaperSelection</Prelink>
        <Postlink>PcChair/A612:SuggestSessions</Postlink>
        <Due>30/11/2000</Due>
        <Description>Match all papers agains keywords defined
        by the conference</Description>
        <Code>http://validation.finge.com/pcchair/a611.html</Code>
  </Activity>
  <Activity>
        <Name>A612:SuggestSessions</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A611:MatchPapers</Prelink>
        <Postlink>PcChair/A613:SetUpSessionCommittees</Postlink>
        <Due>3/12/2000</Due>
        <Description>Suggest a session division according to subjects
        and create a preliminary Conference-program</Description>
        <Code>http://validation.finge.com/pcchair/a612.html</Code>
  </Activity>
  <Activity>
        <Name>A613:SetUpSessionCommittees</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>PcChair/A612:SuggestSessions</Prelink>
        <Postlink>SessionChair[1-5]/A621:DetermineSessionSubjectAndGoal</Postlink>
        <Due>5/12/2000</Due>
        <Description>Choose Session Chairs (from PC Members) for every
        session and notify them</Description>
        <Code>http://validation.finge.com/pcchair/a613.html</Code>
  </Activity>
</ProcessFragment>

<ProcessFragment>
  <Name>SelectPapersAndPlanSessions</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A621:DetermineSessionSubjectAndGoal</Name>
        <Workspace>SessionChair[1-5]</Workspace>
        <Prelink>PcChair/A613:SetUpSessionCommittees</Prelink>
        <Postlink>SessionChair[1-5]/A622:CheckPapersForSession</Postlink>
        <Due>10/12/2000</Due>
        <Description>Write an initial Session description</Description>
        <Code>http://validation.finge.com/sessionchair/a621.html</Code>
  </Activity>
  <Activity>
        <Name>A622:CheckPapersForSession</Name>
        <Workspace>SessionChair[1-5]</Workspace>
        <Prelink>SessionChair[1-5]/A621:DetermineSessionSubjectAndGoal</Prelink>
        <Postlink>SessionChair[1-5]/A623:PaperAllocation</Postlink>
        <Due>15/12/2000</Due>
        <Description>Select papers that should be
        included in your session</Description>
        <Code>http://validation.finge.com/glueserver.cgi?
        processfragment=SessionChair[1-5]/A622:CheckPapersForSession&
        agentclass=agent.Selection.Session</Code>
  </Activity>
  <Activity>
        <Name>A623:PaperAllocation</Name>
        <Workspace>SessionChair[1-5]</Workspace>
        <Prelink>SessionChair[1-5]/A622:CheckPapersForSession</Prelink>
        <Postlink>SessionChair[1-5]/A624:CheckTimeslotForSession</Postlink>
        <Due>15/12/2000</Due>
        <Description>Allocate papers to a session</Description>
        <Code>http://validation.finge.com/glueserver.cgi?
        processfragment=SessionChair[1-5]/A623:PaperAllocation&
        agentclass=agent.Negotiation.Session</Code>
  </Activity>
  <Activity>
        <Name>A624:CheckTimeslotForSession</Name>
        <Workspace>SessionChair[1-5]</Workspace>
        <Prelink>SessionChair[1-5]/A623:PaperAllocation</Prelink>
        <Postlink>SessionChair[1-5]/A625:SessionAllocation</Postlink>
        <Due>18/12/2000</Due>
        <Description>Select two timeslots for your session</Description>
        <Code>http://validation.finge.com/glueserver.cgi?
        processfragment=SessionChair[1-5]/A624:CheckTimeslotForSession&
        agentclass=agent.Selection.Timeslot</Code>
  </Activity>
  <Activity>
        <Name>A625:SessionAllocation</Name>
        <Workspace>SessionChair[1-5]</Workspace>
        <Prelink>SessionChair[1-5]/A624:CheckTimeslotsForSession</Prelink>
        <Postlink>SessionChair[1-5]/A626:PublishSessionDescription</Postlink>
        <Due>19/12/2000</Due>
        <Description>Allocate session to timeslots</Description>
        <Code>http://validation.finge.com/glueserver.cgi?
        processfragment=A625:SessionAllocation&
        agentclass=agent.Negotiation.Timeslot</Code>
  </Activity>
  <Activity>
        <Name>A626:PublishSessionDescription</Name>
        <Workspace>SessionChair[1-5]</Workspace>
        <Prelink>SessionChair[1-5]/A625:SessionAllocation</Prelink>
        <Postlink>PcChair/A7:PublishConferenceProgram</Postlink>
        <Due>20/12/2000</Due>
        <Description>Update and publish Session-description</Description>
        <Code>http://validation.finge.com/sessionchair/a626.html</Code>
  </Activity>
</ProcessFragment>

<ProcessFragment>
  <Name>PublishConferenceProgram</Name>
  <Workspace>Conference</Workspace>
  <Activity>
        <Name>A7:PublishConferenceProgram</Name>
        <Workspace>PcChair</Workspace>
        <Prelink>SessionChair[1-5]/A626:PublishSessionDescription</Prelink>
        <Due>15/01/2001</Due>
        <Description>Update and distribute the final
        Conference-program</Description>
        <Code>http://validation.finge.com/pcchair/a7.html</Code>
  </Activity>
</ProcessFragment>

</Process>
```

H.2 GlueModel for the Scenario

Here is the GlueModel for the conference organising scenario, defining the relationships between process fragments (listed in section H.1) and software agents (listed in section H.3).

```
<GlueModel>
<fragment-agent-pair>
  <agent agent-class="agent.Selection.Paper" amp-id="ConferenceAMP">
        <interaction-type>coordination</interaction-type>
        <result>ok|fail</result>
  </agent>
  <fragment fragment-id="PcMember[1-25]/A3.1:SelectPaper">
    <reaction>
        <result>ok</result>
        <action fragment-id="PcMember[1-25]/A3.2:ReviewerAllocation"
        body="execute_process_fragment_PFNUMBER">
        </action>
        <result>fail</result>
        <action fragment-id"PcMember[1-25]/A3.1:SelectPaper"
        body="execute_process_fragment_PFNUMBER">
    </reaction>
  </fragment>
</fragment-agent-pair>

<fragment-agent-pair>
  <agent agent-class="agent.Negotiation.Reviewer" amp-id="ConferenceAMP">
        <interaction-type>negotiation</interaction-type>
        <result>ok|select</result>
  </agent>
  <fragment fragment-id="PcMember[1-25]/A3.2:ReviewerAllocation">
    <reaction>
        <result>ok</result>
        <action fragment-id="PcMember[1-25]/4.1:ViewPaper"
        body="execute_process_fragment_PFNUMBER">
        </action>
        <result>select</result>
        <action fragment-id"PcMember[1-25]/A3.1:SelectPaper"
        body="execute_process_fragment_PFNUMBER"></action>
    </reaction>
  </fragment>
</fragment-agent-pair>

<fragment-agent-pair>
  <agent agent-class="agent.Selection.Session" amp-id="ConferenceAMP">
        <interaction-type>coordination</interaction-type>
        <result>ok|fail</result>
  </agent>
  <fragment fragment-id="SessionChair[1-5]/A6.2.2:CheckPaperForSession">
    <reaction>
        <result>ok</result>
        <action fragment-id="SessionChair[1-5]/A6.2.3:PaperAllocation"
        body="execute_process_fragment_PFNUMBER"></action>
        <result>fail</result>
        <action fragment-id="SessionChair[1-5]/A6.2.2:CheckPaperForSession"
        body="execute_process_fragment_PFNUMBER"></action>
    </reaction>
  </fragment>
</fragment-agent-pair>

<fragment-agent-pair>
  <agent agent-class="agent.Negotiation.Session" amp-id="ConferenceAMP">
        <interaction-type>negotiation</interaction-type>
        <result>ok|fail</result>
  </agent>
  <fragment fragment-id="SessionChair[1-5]/A6.2.3:PaperAllocation">
    <reaction>
        <result>ok</result>
        <action fragment-id="SessionChair[1-5]/A6.2.4:CheckTimeslotForSession"
        body="execute_process_fragment_PFNUMBER"></action>
        <result>fail</result>
        <action fragment-id="SessionChair[1-5]/A6.2.3:PaperAllocation"
        body="execute_process_fragment_PFNUMBER"></action>
    </reaction>
  </fragment>
</fragment-agent-pair>

<fragment-agent-pair>
  <agent agent-class="agent.Selection.Timeslot" amp-id="ConferenceAMP">
        <interaction-type>coordination</interaction-type>
        <result>ok|fail</result>
  </agent>
  <fragment fragment-id="SessionChair[1-5]/A6.2.4:CheckTimeslotForSession">
    <reaction>
        <result>ok</result>
        <action fragment-id="SessionChair[1-5]/A6.2.5:SessionAllocation"
        body="execute_process_fragment_PFNUMBER"></action>
        <result>fail</result>
        <action fragment-id="SessionChair[1-5]/A.2.4:CheckTimeslotForSession"
        body="execute_process_fragment_PFNUMBER"></action>
    </reaction>
  </fragment>
</fragment-agent-pair>

<fragment-agent-pair>
  <agent agent-class="agent.Negotiation.Timeslot" amp-id="ConferenceAMP">
        <interaction-type>negotiation</interaction-type>
        <result>ok|fail</result>
  </agent>
  <fragment fragment-id="SessionChair[1-5]/A6.2.5:SessionAllocation">
    <reaction>
        <result>ok</result>
        <action fragment-id="SessionChair[1-5]/A6.2.6:PublishSessionDescription"
        body="execute_process_fragment_PFNUMBER"></action>
        <result>fail</result>
        <action fragment-id="SessionChair[1-5]/A6.2.5:SessionAllocation"
        body="execute_process_fragment_PFNUMBER"></action>
    </reaction>
  </fragment>
</fragment-agent-pair>

</GlueModel>
```

H.3 CAGIS Cooperative Agents Support for the Scenario

This section contains the Java source code for the agents used to represent the two cooperative activities *A6.2.4:Check timeslot for session* and *A6.2.5:Session allocation* (section 20.5.6).

H.3.1 Session Selection Agent

Here is the Java code for the Session Selection Agent.

```
package no.ntnu.diplom.userAgents;

import no.ntnu.diplom.systemAgents.*;
import net.jini.core.entry.Entry;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;


/**
* This is the Session Selection Agent. This agent is distributed to all
* the Session Chairs of the Program Committee during the Select Papers
* and Plan Sessions activity. The agent provides a graphical user interface
* (GUI) for the users. The agent collects the result from the users and returns
* home.
*/
public class SessionSelectionAgent extends UserAgent implements ActionListener{

      public Boolean finished = new Boolean(false);
      public Vector route = new Vector();
      public Vector selection = new Vector();
      public JFrame frame = new JFrame ("User Agent");
      public Button send_button = new Button("Send");
      public Button start_button = new Button("Start");
      public Button dispose_button = new Button("Dispose");
      public Button ok_button = new Button("Ok");
      public JLabel name_text = new JLabel("Session Selection Agent");
      public JTextArea locationArea = new JTextArea(30,40);
      public JTextArea description_text;
      public String[] columnNames = {"Id","Author",
                  "Title",
                  "Select"};
      public Object[][] data;
      public JTable tab;
      public String posReviewer;

      //Constructors

      public SessionSelectionAgent(){
              agentType = "UserAgent";}



        /**
         * This method starts the agent at the different Agent Places. When
         * the mobile agent is taken out of the JavaSpace, it needs to be
         * activated. The Receiver Agent takes care of this by invoking this
         * method. (The name of the "activate method" is standard for all the
         * mobile agents in this architecture.)
         */
      public void startAgent(){
              if(finished.booleanValue()){
                      finish();}
              else{
                      String location = destinationID;
                      posReviewer = destinationID;

                      description_text = new JTextArea(" Hello member "
                               + location + " (Session Chair), \n"
                               + " please select desired papers from the table.");


                      MyTableModel myModel = new MyTableModel(columnNames, data);
                      JTable table = new JTable(myModel);
                      tab = new JTable();
                      tab = table;

                      TableColumn column0 = table.getColumnModel().getColumn(0);
                      column0.setPreferredWidth(15);
                      TableColumn column1 = table.getColumnModel().getColumn(1);
                      column1.setPreferredWidth(150);
                      TableColumn column2 = table.getColumnModel().getColumn(2);
                      column2.setPreferredWidth(250);
                      TableColumn column3 = table.getColumnModel().getColumn(3);
                      column3.setPreferredWidth(5);

                      //Panel for the table
                      JPanel tablePanel = new JPanel();
                      tablePanel.setLayout(new GridLayout(0, 1));
                      tablePanel.add(table);
                      tablePanel.setBounds(20,130,550,150);

                      //Scrollbar for the table
                      JScrollPane scrollPaneTable = new JScrollPane(table);
                      tablePanel.setBorder(BorderFactory.createLineBorder(Color.black));
                      tablePanel.add(scrollPaneTable, BorderLayout.CENTER);
                      scrollPaneTable.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

                      frame.getContentPane().setLayout(null);
                      description_text.setEditable(false);
                      description_text.setBackground(frame.getBackground());

                      description_text.setBounds(170,70,350,55);
                      description_text.setFont(new Font("Serif", Font.ITALIC, 16));

                      name_text.setFont(new Font("Comics", Font.BOLD, 20));
                      name_text.setBounds(200,20,250,30);
                      send_button.setBounds(250,310,70,30);
                      frame.setBounds(400,100,600,400);
                      send_button.addActionListener(this);

                      locationArea.setBounds(170,155,130,30);
                      locationArea.append("Location: " + location);
                      locationArea.setLineWrap(true);
                      locationArea.setEditable(false);
                      Color color = frame.getBackground();
                      locationArea.setBackground(color);

                      if(!route.isEmpty()){
                              destinationID = (String)route.remove(0);}
                      else{
                              destinationID = "amp";}

                      frame.getContentPane().add(name_text);
                      frame.getContentPane().add(description_text);
                      frame.getContentPane().add(send_button);
                      frame.getContentPane().add(tablePanel);

                      frame.setResizable(false);
                      frame.show();
              }
      }
```

```
/**
 * This method is called when the Session Selection Agent is initiated at
 * the AMP. The init() method gets the needed information (Session Chairs
 * and papers) from the repository by using the Repository Agent.
 * The method sends one instance of the Session Selection Agent to each of
 * the Session Chairs in the Program Committee. This way the members can
 * select possible papers for their sessions in papallel.
 */
public void init(){
        RepositoryAgent ra = new RepositoryAgent();
        route = ra.getSessionChairs();
        data = ra.getPapers();

        SessionSelectionAgent nextAgent;
        DispatcherAgent da = new DispatcherAgent();

        for (Enumeration e = route.elements() ; e.hasMoreElements() ;) {
                nextAgent = new SessionSelectionAgent();
                nextAgent.destinationID = (String)e.nextElement();
                nextAgent.data = data;

                //Tell the DispatcherAgent to dispatch the current Agent
                da.dispatchMe(nextAgent, "conference", nextAgent.destinationID);
        }
}



  /**
   * This method is called if the agent is retrieved manually from the
   * space without having done any work.
   */
public void disposeAgent(){
        frame.getContentPane().setLayout(null);
        name_text.setBounds(85,20,200,30);
        dispose_button.setBounds(110,120,70,30);
        frame.setBounds(400,100,300,200);
        dispose_button.addActionListener(this);

        frame.getContentPane().add(name_text);
        frame.getContentPane().add(dispose_button);
        frame.setResizable(false);
        frame.show();
}


  /**
   * This method is invoked when the mobile agent successfully returns from
   * its trip. The method updates the repository with the collected information
   * by using the Repository Agent.
   */
        public void finish(){
        RepositoryAgent ra = new RepositoryAgent();

        JTextArea finished_text = new JTextArea(" The agent has returned\n"
                          +" successfully from member "
                          + (String)selection.elementAt(0) + "!");

        ra.insertSessionPapers(selection);

        finished_text.setEditable(false);
        finished_text.setBackground(frame.getBackground());

        frame.getContentPane().setLayout(null);
        name_text.setBounds(85,20,200,30);
        finished_text.setBounds(70,60,200,50);

        ok_button.setBounds(110,120,70,30);
        frame.setBounds(400,100,300,200);
        ok_button.addActionListener(this);

        frame.getContentPane().add(name_text);
        frame.getContentPane().add(finished_text);
        frame.getContentPane().add(ok_button);
        frame.setResizable(false);
        frame.show();
}



  /**
   * This method is invoked when either the "Send" button, "Dispose"
   * button or the "Ok" button is presed. The method performs the
   * necessary actions.
   *
   * @param   ae  This is the ActionEvent causing the invokation.
   */
public void actionPerformed(ActionEvent ae){

        if (ae.getSource() == send_button){
                selection = new Vector();
```

```
                        selection.addElement(posReviewer);
                        int num = tab.getRowCount();
                        for(int i=0; i<num; i++){
                          String bol = (String)tab.getValueAt(i, 3).toString();
                          if(bol.equals("true")){
                                  selection.addElement((String)tab.getValueAt(i, 0).toString());
                          }
                        }

                      SessionSelectionAgent nextAgent = new SessionSelectionAgent();
                      destinationID = "amp";
                      nextAgent.finished = new Boolean(true);
                      nextAgent.data = data;
                      nextAgent.selection = selection;

                      //Tell the DispatcherAgent to dispatch the current Agent
                      DispatcherAgent da = new DispatcherAgent();
                      da.dispatchMe(nextAgent, "conference", destinationID);
                      frame.dispose();
              }

              if (ae.getSource() == dispose_button){
                frame.dispose();}

              if (ae.getSource() == ok_button){
                frame.dispose();}
      }//end_actionPerformed
}
```

H.3.2 Session Allocation Agent

Here is the Java-code for the session allocation agent.

```
package no.ntnu.diplom.negotiationAgents;

import no.ntnu.diplom.systemAgents.*;

import net.jini.core.entry.Entry;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.border.*;
import java.awt.event.*;
import java.awt.*;
import java.util.*;


/**
* This is the Session Allocation Agent. If more than one Session Chair
* are interested in the same paper for their session, the Session Allocation
* Agent will take care of the negotiation so that the number of sessions
* for each paper don’t exceeds one. The Session Allocation Agent is a mobile
* Negotiation Agent which can migrate to the Session Chairs involved in the
* conflict in order to solve it. The agent provides a graphical user interface
* (GUI) for the users. It collects the result from the users and returns home.
*/
public class SessionAllocationAgent extends NegotiationAgent implements ActionListener{

      public Boolean finished = new Boolean(false);
      public Boolean removeMessage = new Boolean(false);
      public Boolean removeRandomMessage = new Boolean(false);
      public Boolean step2_finished = new Boolean(false);
      public Boolean step2 = new Boolean(false);
      public Boolean step2_nextround = new Boolean(false);
      public Vector route = new Vector();
      public Vector selection = new Vector();
      public JFrame frame = new JFrame ("Negotiation Agent");
      public Button send_button = new Button("Send");
      public Button dispose_button = new Button("Dispose");
      public Button ok_button = new Button("Ok");
      public Button yes_button = new Button("Yes");
      public Button no_button = new Button("No");
      public JLabel name_text = new JLabel("Session Allocation Agent");
      public JTextArea locationArea = new JTextArea(30,40);
      public JTextArea description_text = new JTextArea("Here comes a little description");

      public Vector papers_Id = new Vector();
      public Vector papers_Prl = new Vector();
      public Vector copyOf_papers_Id = new Vector();
      public Vector copyOf_papers_Prl = new Vector();
      public Vector conflictMembers = new Vector();
      public Vector removedPerson = new Vector();
      public Vector removedFrom = new Vector();
      public Object[][] data;
      public String paperNumber;

      //Constructors

      public SessionAllocationAgent(){
              agentType = "NegotiationAgent";
      }

         /**
         * This method activates the agent at the different Agent Places. When
         * the mobile agent is taken out of the JavaSpace, it needs to be
         * activated. The Receiver Agent takes care of this by invoking this
         * method. (The name of the "activate method" is standard for all the
         * mobile agents in this architecture.)
         */
      public void startAgent(){
              if(finished.booleanValue()){}
              else if(removeMessage.booleanValue()){
                      frame = new JFrame("Negotiation Agent - Message");
                      frame.getContentPane().setLayout(null);
                      name_text.setFont(new Font("Comics", Font.BOLD, 18));
                      name_text.setBounds(80,20,250,30);
                      ok_button.setBounds(155,160,70,30);
                      frame.setBounds(400,100,400,230);
                      description_text = new JTextArea("Hello member " + destinationID + " (Session Chair).\n"
                                  + "Paper " + paperNumber + " is in conflict. Since you are one of the\n"
                                  +       "Session Chairs who have selected most papers,\n"
                                  +       "you have been removed from paper " + paperNumber + ".");
                      description_text.setEditable(false);
                      description_text.setBackground(frame.getBackground());
                      description_text.setBounds(70,60,350,90);
                      description_text.setFont(new Font("Serif", Font.PLAIN, 14));

                      ok_button.addActionListener(this);

                      frame.getContentPane().add(name_text);
                      frame.getContentPane().add(description_text);
                      frame.getContentPane().add(ok_button);
                      frame.setResizable(false);
                      frame.show();
              }
              else if(removeRandomMessage.booleanValue()){
                      frame = new JFrame("Negotiation Agent - Message");
                      frame.getContentPane().setLayout(null);
                      name_text.setFont(new Font("Comics", Font.BOLD, 18));
                      name_text.setBounds(80,20,250,30);
                      ok_button.setBounds(155,130,70,30);
                      frame.setBounds(400,100,400,200);
                      description_text = new JTextArea("Hello member " + destinationID + " (Session Chair).\n"
                              + "You have been removed at random from paper " + paperNumber);
                      description_text.setEditable(false);
                      description_text.setBackground(frame.getBackground());
                      description_text.setBounds(70,60,350,50);
                      description_text.setFont(new Font("Serif", Font.PLAIN, 14));

                      ok_button.addActionListener(this);

                      frame.getContentPane().add(name_text);
                      frame.getContentPane().add(description_text);
                      frame.getContentPane().add(ok_button);
                      frame.setResizable(false);
                      frame.show();
              }
              else if(step2.booleanValue()){
                      frame = new JFrame("Negotiation Agent - Paperconflict");
                      frame.getContentPane().setLayout(null);
                      name_text.setFont(new Font("Comics", Font.BOLD, 18));
                      name_text.setBounds(80,20,250,30);
                      yes_button.setBounds(115,160,70,30);
                      no_button.setBounds(200,160,70,30);
                      frame.setBounds(400,100,400,230);
                      description_text = new JTextArea("Hello member " + destinationID + ".\n"
                              +       "Paper " + paperNumber + " is in conflict.\n"
                              + "Are you willing to remove it from your session list?");
                      description_text.setEditable(false);
                      description_text.setBackground(frame.getBackground());
                      description_text.setBounds(70,60,350,70);
                      description_text.setFont(new Font("Serif", Font.PLAIN, 14));

                      yes_button.addActionListener(this);
                      no_button.addActionListener(this);

                      frame.getContentPane().add(name_text);
                      frame.getContentPane().add(description_text);
                      frame.getContentPane().add(yes_button);
                      frame.getContentPane().add(no_button);
                      frame.setResizable(false);
                      frame.show();
              }
              else if(step2_nextround.booleanValue()){
                      System.out.println("Step 2. going for the next paper");
                      handleConflictStep2();
              }

              else if(step2_finished.booleanValue()){
                      //Step3
                      System.out.println("Step 2 finished.");

                      for(int i=0; i<removedPerson.size(); i++){
                              System.out.println("Removing member " + removedPerson.elementAt(i)
                                         + " from paper " + removedFrom.elementAt(i));
                              RepositoryAgent ra = new RepositoryAgent();
                              ra.removeSession((String)removedPerson.elementAt(i),
                                         (String)removedFrom.elementAt(i));
                      }

                      Vector[] papers = findConflictPapers();
                      papers_Id = papers[0];
                      papers_Prl = papers[1];

                      if(papers_Id.size() > 0){
                              System.out.println("Step2 didn't remove all conflicts. Start step3.");
                              handleConflictStep3();
                      }
              }

      }


      /**
       * This method is called when the Session Allocation Agent is initiated at
       * the AMP. The init() method gets the needed information (Session Chairs and
       * papers) from the repository by using the Repository Agent.
       * The method collects the different papers and starts the negotiaton
       * process if some of the papers are in conflict.
       */
      public void init(){
              RepositoryAgent ra = new RepositoryAgent();

              Vector[] papers = findConflictPapers();
              papers_Id = papers[0];
              papers_Prl = papers[1];

              if(papers_Id.size() > 0){
                      handleConflictStep1();}

              if(papers_Id.size() > 0){
                      copyOf_papers_Id = papers_Id;
                      copyOf_papers_Prl = papers_Prl;
                      handleConflictStep2();
              }
      }


        /**
         * This method sends a message to a given Session Chair of the Program Committee
         * if the Session Chair is randomly removed from a paper during step 3 of the
         * negotiation process.
         *
         * @param   paperId  The idetifier for the paper the member is removed from.
         * @param   member  The idetifier for the removed member.
         */
         public void sendRemovedMessage(String paperId, String member, Boolean random){
                      SessionAllocationAgent nextAgent;
                      DispatcherAgent da = new DispatcherAgent();

                      nextAgent = new SessionAllocationAgent();
                      nextAgent.destinationID = member;
                      nextAgent.agentType = null;
                      nextAgent.paperNumber = paperId;
                      if(random.booleanValue()){
                              nextAgent.removeRandomMessage = new Boolean(true);}
                      else{
                              nextAgent.removeMessage = new Boolean(true);}
                      nextAgent.papers_Id = papers_Id;
                      nextAgent.papers_Prl = papers_Prl;

                      //Tell the DispatcherAgent to dispatch the current Agent
                      da.dispatchMe(nextAgent, "conference", nextAgent.destinationID);
                      frame.dispose();
         }


        /**
         * This method gets all the possible session papers from the repository
         * by using the Repository Agent. The papers which have exactly one
         * possible session are updated to the repostory as actual session (this
         * session are allocated to the paper). The papers which have more than
         * one possible session registered to them are sorted in different
         * vectors and returned.
         *
         * @return     An array of vectors with the papers in conflict.
```

```
        */
        public Vector[] findConflictPapers(){

             Vector papers_Id = new Vector();
             Vector papers_Prl = new Vector();

             RepositoryAgent ra = new RepositoryAgent();
             data = ra.getPossibleSessionPapers();

             for (int i = 0; i < data.length; i++) {
                     for (int j = 1; j < data[i].length; j++) {

                             if(numberOfPossibleSessions((String)data[i][1]) == 1){
                                     Vector readyPapers = new Vector();

                                     /*Format for the vector (papers):
                                      * 0 - id
                                      * 1 - session list
                                      */
                                      readyPapers.addElement(data[i][0]);
                                      readyPapers.addElement(data[i][1]);
                                      ra.allocateSession(readyPapers);
                             }

                             if(numberOfPossibleSessions((String)data[i][1]) > 1){
                                      papers_Id.addElement(data[i][0]);
                                      papers_Prl.addElement(data[i][1]);
                             }
                     }
             }
           Vector[] papers = {papers_Id,papers_Prl};
           return papers;
        }


       /**
        * This is step 1 of the negotiation process. This method are used during this
        * step to remove the Session Chairs who have selected the most papers for their session
        * from the given paper, and send out a message to the removed Session Chairs.
        * If some of the papers get exactly one session during this step, the Session Chair
        * are allocated as actual session for the papers, and the repository is updated (through the
        * Repository Agent).
        */
        public void handleConflictStep1(){
             System.out.println("Step 1 begins...");
             Vector psl = new Vector();
             Vector internal_removedMembers = new Vector();
             Vector internal_removedFrom= new Vector();

             //Remove given person from given paper!!!
             //Only the papers in conflict (papers_prl) are considered.
             for(int i=0; i<papers_Id.size(); i++){
                     int personId = findBiggestSelectionActor(papers_Prl);
                     psl = new Vector();
                     if(personId != -1 && papers_Prl.size() > i){
                             String dummy = (String)papers_Prl.remove(i)+",";
                             while(dummy.length()>0){
                                     Integer num =
                                     new Integer(Integer.parseInt(dummy.substring(0, dummy.indexOf(","))));
                                     psl.addElement(num);
                                     dummy = dummy.substring(dummy.indexOf(",") + 1);
                             }
                             for(int j=0; j<psl.size(); j++){
                                     if((Integer.toString(personId)).equals(psl.elementAt(j).toString())){
                                             String removed = psl.remove(j).toString();

                                             //Save the removed members and paperIds
                                             internal_removedMembers.addElement(removed);
                                             internal_removedFrom.addElement(papers_Id.elementAt(i));
                                     }
                             }

                             if(psl.size() > 1){
                                     String pslString = new String();
                                     for(int j=0;j<psl.size();j++){
                                             //Formatting the String
                                             if(j == 0){
                                                     pslString = pslString + (psl.elementAt(j)).toString();}
                                             else{
                                                     pslString = pslString + "," + (psl.elementAt(j)).toString();}
                                     }
                                     papers_Prl.add(i,pslString);
                             }//end_if
                     }
             }//end_for

             for(int k=0; k<internal_removedMembers.size(); k++){
                             System.out.println("Message to member#: " + internal_removedMembers.elementAt(k)
                                     + " - Removed from: "   + internal_removedFrom.elementAt(k));
                             RepositoryAgent ra = new RepositoryAgent();
                             ra.removeSession((String)internal_removedMembers.elementAt(k),
                                                  (String)internal_removedFrom.elementAt(k));
```

```
                }

                Vector[] papers = findConflictPapers();
                papers_Id = papers[0];
                papers_Prl = papers[1];

                SessionAllocationAgent nextAgent;
                DispatcherAgent da = new DispatcherAgent();

                for (int h=0; h<internal_removedMembers.size(); h++) {
                                removedPerson.addElement(internal_removedMembers.elementAt(h));
                                removedFrom.addElement(internal_removedFrom.elementAt(h));

                                sendRemovedMessage((String)internal_removedFrom.elementAt(h),
                                  (String)internal_removedMembers.elementAt(h),
                                new Boolean(false));
                }
          }

          /**
           * The method finds the next stop of its journey and starts the
           * migration of the agent to the destination. This is step 2 of
           * the negotiation process. During this step the Session Allocation
           * Agent migrates to the Sesion Chairs involved in the overbooked papers.
           */
           public void handleConflictStep2(){
                System.out.println("Step2 begins...=================================================");

                String paper = (String)copyOf_papers_Id.remove(0);
                String temproute = (String)copyOf_papers_Prl.remove(0) + ","; //Special format
                conflictMembers = new Vector();

                while(temproute.length()>0){
                        Integer num = new Integer(Integer.parseInt(temproute.substring(0,
                                      temproute.indexOf(","))));
                        conflictMembers.addElement(num);
                        route.addElement(num);
                        temproute = temproute.substring(temproute.indexOf(",") + 1);
                }


                SessionAllocationAgent nextAgent;
                DispatcherAgent da = new DispatcherAgent();

                nextAgent = new SessionAllocationAgent();
                nextAgent.destinationID = route.remove(0).toString();
                nextAgent.route = route;
                nextAgent.paperNumber = paper;
                nextAgent.step2 = new Boolean(true);
                nextAgent.papers_Id = papers_Id;
                nextAgent.papers_Prl = papers_Prl;
                nextAgent.copyOf_papers_Id = copyOf_papers_Id;
                nextAgent.copyOf_papers_Prl = copyOf_papers_Prl;
                nextAgent.conflictMembers = conflictMembers;
                nextAgent.removedPerson = removedPerson;
                nextAgent.removedFrom = removedFrom;

                da.dispatchMe(nextAgent, "conference", nextAgent.destinationID);
                frame.dispose();
        }


          /**
           * This method removes a possible Session Chair from a paper at random,
           * updates the repository (using the findConflictPapers() method
           * which again uses the Repository Agent) and sends a message to
           * the removed Session Chair.
           */
           public void handleConflictStep3(){
                System.out.println("Step3 begins...=================================================");

                while(papers_Id.size() > 0){
                        System.out.println("In conflict" + " - " + papers_Prl.elementAt(0));

                        int member = findRandomMember((String)papers_Prl.elementAt(0));
                        System.out.println("Random member " + member + " removed from "
                                   + papers_Id.elementAt(0));
                        RepositoryAgent ra = new RepositoryAgent();
                        ra.removeSession(Integer.toString(member), (String)papers_Id.elementAt(0));

                        sendRemovedMessage((String)papers_Id.remove(0),Integer.toString(member),
                                  new Boolean(true));
                        Vector[] papers = findConflictPapers();
                        papers_Id = papers[0];
                        papers_Prl = papers[1];
                }
          }


          /**
           * This method is called if the agent is retrieved manually from the
           * space without having done any work.
```

```
         */
         public void disposeAgent(){
              frame.getContentPane().setLayout(null);
              name_text.setBounds(85,20,200,30);
              dispose_button.setBounds(110,120,70,30);
              frame.setBounds(400,100,300,200);
              dispose_button.addActionListener(this);

              frame.getContentPane().add(name_text);
              frame.getContentPane().add(dispose_button);
              frame.setResizable(false);
              frame.show();
        }


        /**
         * This method finds the number of possible sessions from a string. The
         * sessions in the string are separated by comma (",").
         *
         * @param   prl  The string with the possible sessions.
         * @return     The number of possible sessions contained in the string.
         */
         public int numberOfPossibleSessions(String prl){
              String workString = prl;
              int number = 1;
              boolean charachterExists = false;
              for(int i=0; i<prl.length();i++){
                if(workString.startsWith(",")){
                      number++; }
                else if(workString.startsWith("NA")){
                      number = 0;}
                workString = prl.substring(i);
              }
              return number;
        }


        /**
         * This method is invoked when either the "Ok" button, the "Yes" button or
         * the "No" button is pressed. The method performs the necessary actions.
         *
         * @param   ae  This is the ActionEvent causing the invokation.
         */
         public void actionPerformed(ActionEvent ae){
              if (ae.getSource() == dispose_button){
                      frame.dispose();}
              if (ae.getSource() == ok_button){
                      frame.dispose();}
              if (ae.getSource() == yes_button){
                      removedPerson.addElement(destinationID);
                      removedFrom.addElement(paperNumber);

                      if(conflictMembers.size() > 0){
                        Integer temp = (Integer)conflictMembers.remove(0);}

                      SessionAllocationAgent nextAgent;
                      DispatcherAgent da = new DispatcherAgent();
                      nextAgent = new SessionAllocationAgent();
                      if(route.size() > 0 && conflictMembers.size() > 1){
                              nextAgent.destinationID = route.remove(0).toString();
                              nextAgent.step2 = new Boolean(true);
                              nextAgent.route = route;
                      }
                      //No more papers....
                      else if(copyOf_papers_Id.size() == 0){
                              nextAgent.step2_finished = new Boolean(true);
                              nextAgent.destinationID = "amp";
                      }
                      //No conflict with this paper....
                      else if(conflictMembers.size() == 1){
                              System.out.println("No more confl. with paper: " + paperNumber);
                              nextAgent.destinationID = "amp";
                              nextAgent.step2_nextround = new Boolean(true);
                      }
                      else{
                              nextAgent.destinationID = "amp";
                              System.out.println("XXX No more confl. with paper: " + paperNumber);
                              nextAgent.step2_nextround = new Boolean(true);
                      }

                      nextAgent.paperNumber = paperNumber;
                      nextAgent.papers_Id = papers_Id;
                      nextAgent.papers_Prl = papers_Prl;
                      nextAgent.copyOf_papers_Id = copyOf_papers_Id;
                      nextAgent.copyOf_papers_Prl = copyOf_papers_Prl;
                      nextAgent.removedPerson = removedPerson;
                      nextAgent.removedFrom = removedFrom;

                      da.dispatchMe(nextAgent, "conference", nextAgent.destinationID);
                      frame.dispose();
            }
            if (ae.getSource() == no_button){
```

```
                        //Moved on without editing the paperlist.
                        SessionAllocationAgent nextAgent;
                        DispatcherAgent da = new DispatcherAgent();
                        nextAgent = new SessionAllocationAgent();
                        if(route.size() > 0){
                                nextAgent.destinationID = route.remove(0).toString();
                                nextAgent.step2 = new Boolean(true);
                                nextAgent.route = route;
                        }
                        else if(copyOf_papers_Id.size() == 0){
                                nextAgent.destinationID = "amp";
                                nextAgent.step2_finished = new Boolean(true);
                        }
                        else{
                                nextAgent.destinationID = "amp";
                                nextAgent.step2_nextround = new Boolean(true);
                        }
                        nextAgent.paperNumber = paperNumber;
                        nextAgent.papers_Id = papers_Id;
                        nextAgent.papers_Prl = papers_Prl;
                        nextAgent.copyOf_papers_Id = copyOf_papers_Id;
                        nextAgent.copyOf_papers_Prl = copyOf_papers_Prl;
                        nextAgent.conflictMembers = conflictMembers;
                        nextAgent.removedPerson = removedPerson;
                        nextAgent.removedFrom = removedFrom;

                        da.dispatchMe(nextAgent, "conference", nextAgent.destinationID);
                        frame.dispose();
                }
        }//end_actionPerformed
}
```

Part V

Bibliography

Bibliography

[ABE00] Anders Andersen, Gordon S. Blair, and Frank Eliassen. OOPP: A reflective component-based middleware. In *Norsk Informatikkonferanse (NIK'2000)*, pages 7–18, Bodø, Norway, November 20-22 2000.

[ACK$^+$94] R. Andersen, R. Conradi, J. Krogstie, G. Sindre, and A. Sølvberg. Project Courses at the NTH: 20 Years of Experience. In J. L. Diaz-Herrera, editor, *Software Engineering Education*, Lecture Notes in Computer Science 750, pages 177–188. Springer Verlag, 1994. ISBN 3-5405-7461-1.

[ACM88] V. Ambriola, P. Ciancarini, and C. Montangero. OIKOS The Architecture of Adaptable Environments for Software Specification and Development. Technical report, Dipartimento di Informatica — Universita di Pisa, October 1988. Draft version.

[Ad 99] Ad Astra Engineering Inc. Jumping Beans - The Mobility Framework. web: http://www.JumpingBeans.com, 1999.

[AIS88] José A. Ambros-Ingerson and Sam Steel. Integrating Planning, Execution and Monitoring. In *Proc. of AAAI'88*, pages 83–88, 1988.

[Alf00] Alf Inge Wang and Anders Aas Hanssen and Bård Smidsrød Nymoen. Design Principles for a Mobile, Multi-Agent Architecture for Cooperative Software Engineering. In *Proc. IASTED Internation Conference Software Engineering and Applications*, pages 134–140, Las Vegas, Nevada, US, 6-9 November 2000.

[Apa95] Gilbert Aparicio. The role of intelligent agents in the information infrastructure. Technical report, IBM, United States, 1995.

[Arn98] Arne Sølvberg. Data and what they refer to. In P. Chen, editor, *Conceptual modeling: Historical perspectives and future trends*, Los Angeles, California, USA, 1998. 16th Int. Conf. on Conceptual modeling.

[B$^+$89] K. Benali et al. Presentation of the ALF Project. In *[MSW90], 23 p.*, May 1989.

[Bak97] Seán Baker. *CORBA Distributed Objects - Using Orbix*. ACM press and Addision-Wesley, 1997. ISBN 0-201-92475-7.

[Bal00] Robert Balzer. Keynote on Current State and Future Perspectives of Software Process Technology. In Reidar Conradi, editor, *Proc. 8th European Software Process Workshop on Software Process Technology (EWSPT'2000)*, page 220, Kaprun (Salzburg), Austria, February 21-25 2000. Springer Verlag LNCS 1780.

[Ban93] Liam J. Bannon. CSCW: An Initial Exploration. *Scandinavian Journal of Information Systems*, 5:3–24, August 1993.

[Bas92] Victor R. Basili. The Experimental Paradigm in Software Engineering. In H. Dieter Rombach, Victor R. Basili, and Richard W. Selby, editors, *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 3–12, Proc. Int'l Workshop, Dagstuhl Castle, Germany, September 14-18 1992. Springer Verlag LNCS 706.

[Bas93] Victor R. Basili. The Experimental Paradigm in Software Engineering. In H.D. Romach, V.R. Basilli, and R.W. Selby, editors, *Experimental Software Engineering Issues: Critical Assessment and Future Directives*, 1993. Springer Verlang, LNCS 706.

[BBFL94] Sergio Bandinelli, Marco Braga, Alfonso Fuggetta, and Luigi Lavazza. The Architecture of the SPADE Process-Centered SEE. In *[War94]*, pages 15–30, 1994.

[BCM$^+$92] Victor R. Basili, G. Caldiera, Frank McGarry, R. Pajerski, G. Page, and S. Waligora. The Software Engineering Laboratory – an Operational Software Experience Factory. In *Proc. 14th Int'l Conference on Software Engineering, Melbourne, Australia*, pages 370–381, May 1992.

[BCN$^+$96] 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.

[BCR94a] Victor R. Basili, Gianluigi Caldeera, and H. Dieter Rombach. *Encyclopedia of Software Engineering*, volume 1, chapter Measurement, pages 646–661. John Wiley Sons, 1994.

[BCR94b] Victor R. Basili, Gianluigi Caldiera, and Hans-Dieter Rombach. The Goal Question Metric Paradigm. In *[Mar94]*, pages 528–532, 1994.

[BDG$^+$94] Alexandros Biliris, Shaul Dar, Narain H. Gehani, H. V. Jagadish, and Krith Ramamritham. Asset: A system for supporting extended transactions. In Richard T. Snodgrass and Marianne Winslett, editors, *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 94)*. ACM Press, May 1994.

[BEM91] Noureddine Belkhatir, Jacky Estublier, and Walcelio L. Melo. Adele2: A Support to Large Software Development Process. In *Proc. 1st Conference on Software Process (ICSP1), Redondo Beach, CA*, pages 159–170, October 1991.

[BEM93] Noureddine Belkhatir, Jacky Estublier, and Walcelio Melo. Software Process Model and Work Space Control in the Adele System. In *[Ost93]*, pages 2–11, 1993.

[Ber96] Philip A. Bernstein. Middleware: A Model for Distributed System Services. *Communications of the ACM*, 39(2):86–98, May 1996.

[BFG93] Sergio Bandinelli, Alfonso Fuggetta, and Carlo Ghezzi. Software Process Model Evolution in the SPADE Environment. *IEEE Trans. on Software Engineering*, pages 1128–1144, December 1993. (special issue on Process Model Evolution).

[BG94] Victor R. Basili and Scott Green. Software Process Evolution at the SEL. *IEEE Software*, pages 58–66, July 1994.

[BHT97] 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.

[Bjø00] Bjørn Haakenstad. GlueServer, support for integrating workflow-systems with interactive agents. Technical report, Norwegian University of Science and Technology (NTNU), March 2000. Technical Report, Dept. of Computer and Information Science, EPOS TR 375, 71p.

[BJJT99] Frances M.T. Brazier, Catholijn M. Jonker, Frederik Jan Jurgen, and Jan Treur. Distributed Scehduling to Support a Call Centre: a Co-operative Multi-Agent Approach. *Applied Artificial Intelligence Journal*, 13:65–90, 1999. Special Issue on Multi-Agent Systems.

[BNF96] Sergio Bandinelli, Elisabetta Di Nitto, and Alfonso Fuggetta. Supporting cooperation in the SPADE-1 environment. *IEEE Transactions on Software Engineering*, 22(2), December 1996.

[BO96] 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.

[Boe81] Barry W. Boehm. *Software engineering economics (on COCOMO model)*. Prentice-Hall, 1981.

[Boe88] Barry W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, pages 61–72, May 1988.

[Boo91] G. Booch. *Object Oriented Design with Applications*. Benjamin/Cummings, California, 1991.

[BR91] Victor R. Basili and Hans D. Rombach. Support for Comprehensive Reuse (on TAME project). *Software Engineering Journal (special issue on Software process and its support)*, 6(5):303–316, September 1991.

[Bre99] Jørgen Andre Brecke. Design and implementation of the Renaissance process system client. Technical report, Dept. of Computer and Information Science, NTNU, Norway, 1999.

[Bro86] Frederick P. Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. In *[Kug86]*, 1986.

[BSK95] 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.

[BvET97] Frances Brazier, Pascal van Eck, and Jan Treur. Modelling Competitive Co-operation of Agents in a Compositional Multi-Agent Framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997. Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems.

[C$^+$88] Reidar Conradi et al., editors. *Norsk Informatikk Konferanse — NIK'88, Sundvolden Turisthotell outside Oslo, 248 p.* Tapir, Trondheim, November 1988.

[C$^+$92] Reidar Conradi et al. Design, Use, and Implementation of SPELL, A Language for Software Process Modeling and Evolution. In *[Der92]*, pages 167–177, 1992.

[C$^+$96] Reidar Conradi et al. CAGIS – Cooperating Agents in the Global Information Space, 38 p. Technical report, IDT, NTNU, June 1996. National Basic Research Project in Distributed Information Systems.

[Cap91] Jones Capers. *Applied Software Measurement: Assuring Productivity and Quality*. Software Engineering Series. McGraw-Hill, 1991.

[Cap98] K. Caputo, editor. *CMM Implementation Guide: Choreographing Software Process Improvement*. Addison Wesley, 1998. ISBN 0-2013-7938-4.

[Car97] Steinar Carlsen. *Conceptual Modeling and Composition of Flexible Workflow Models*. PhD thesis, Dept. of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, December 15 1997.

[CDE$^+$00] Pierre-Yves Cunin, Sami Dami, Jacky Estublier, Gianpaolo Cugola, Alfonso Fuggetta, H. Verjus, F. Pacull, and M. Rivière. Support for Software Federations: The PIE Platform. In Reidar Conradi, editor, *Proc. 8th European Software Process Workshop on Software Process Technology (EWSPT'2000), Kaprun (Salzburg), Austria, 21–25 Feb. 2000*, pages 38–52. Springer Verlag LNCS 1780, February 2000.

[CDW87] Reidar Conradi, Tor M. Didriksen, and Dag H. Wanvik, editors. *Proc. IFIP WG-2.4 International Workshop on Advanced Programming Environments, 16-18 June 1986, Trondheim, Norway*. Springer Verlag *LNCS 244*, 604 p., March 1987.

[CFJ98] Reidar Conradi, Alfonso Fugetta, and Mari Letizia Jaccheri. Six theses on software process research. In Volker Gruhn, editor, *Software Process Technology, 6th European Workshop (EWSPT'98)*, pages 100–104, Weybridge, UK, September 16-18 1998. Springer Verlag LNCS 1487.

[CFM00] Paolo Ciancarini, Francesco Franze, and Cecilia Mascolo. Using a Coordination Language to Specify and Analyze Systems containing Mobile Components. *ACM Trans. on Software Engineering and Methodology*, 9(2):167–198, 2000.

[CG98] Gianpaulo Cugola and Carlo Ghezzi. Software Processes: a Retrospective and a Path to the Future. *SOFTWARE PROCESS – Improvement and Practice*, 4(2):101–123, 1998.

[CHL95] Reidar Conradi, Marianne Hagaseth, and Chunnian Liu. Planning Support for Cooperating Transactions in EPOS. *Information Systems*, 20(4):317–326, June 1995.

[CHLN94] Reidar Conradi, Marianne Hagaseth, Jens-Otto Larsen, and Minh Nguyen. EPOS: Object-Oriented and Cooperative Process Modelling. In *[FKN94]*, pages 33–70, 1994.

[CJM92] Reidar Conradi, M.Letizia Jaccheri, and Cristina Mazzi. Design, Use and Implementation of SPELL, a language for Software Process Modeling and Evolution. In *Proc. Second European Workshop on Software Process Technology (EWSPT'92), Trondheim, Norway.*, pages 196–214, 1992.

[Cla93] Claude Godart. COO: A transaction model to support cooperation software developers coordination. In *4th European Software Engineering Conference*, pages 361–379, Garmisch, Germany, 1993. Springer Verlag, LNCS 717.

[CLH95] Reidar Conradi, Chunnian Liu, and Marianne Hagaseth. Planning Support for Cooperating Transactions in EPOS. *Information Systems*, 20(4):317–326, June 1995.

[CM96] Anthony Chavez and Pattie Maes. Kasbah: An Agent Marketplace for Buying and Selling Goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, April 1996.

[CMM97] Anthony Chavez, Alexandros G. Moukas, and Pattie Maes. Challenger: A Multi-agent System for Distributed Resource Allocation. In *Proceedings of the International Conference on Autonomous Agents*, Marina Del Ray, California, USA, 1997.

[CNF98] Gianpaolo Cugola, Elisabetta Di Nitto, and Alfonso Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *Proc. 20th Int'l Conf. on Software Engineering (ICSE'98)*, Kyoto, Japan, April 1998. IEEE CS Press.

[CNWL98] Reidar Conradi, Minh Ngoc Nguyen, Alf Inge Wang, and Chunnian Liu. Planning Support to Software Process Evolution. In *Proc. Eight International Conference on Software Engineering and Knowledge Engineering (SEKE'98), 18–20 June 1998*, page 16 p., San Francisco, USA, 1998.

[CNWL00] Reidar Conradi, Minh Ngoc Nguyen, Alf Inge Wang, and Chunnian Liu. Planning Support to Software Process Evolution. *International Journal of Software Engineering and Knowledge Engineering, Special Issue: Best papers from SEKE'98*, 10(1):31–37, February 2000.

[Con98] Renaissance Consortium. *The RENAISSANCE Method Handbook*. Published by web, June 1998. http://www.comp.lancs.ac.uk/computing/research/cweg/projects/renaissance.

[Cor00a] Microsoft Corp. Visual Studio home page. web: http://msdn.microsoft.com/vstudio/, November 2000.

[Cor00b] Rational Software Corp. Unified Modeling Language Resource Center. web: http://www.rational.com/uml, 2000.

[Cor00c] Toshiba Corporation. Multi-Agent Framework for 100% Pure Agent System. web: http://www2.toshiba.co.jp/beegent/, 2000.

[COWL91] Reidar Conradi, Espen Osjord, Per H. Westby, and Chunnian Liu. Initial Software Process Management in EPOS. *Software Engineering Journal (Special Issue on Software process and its support)*, 6(5):275–284, September 1991.

[CR94] Panos K. Chrysanthis and Krithi Ramamritham. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems*, 19(3):450–491, sept 1994.

[CRW96] Microsoft Corporation, Redmond, and Washington. Microsoft DCOM: A Technical Overview. web: http://www.eu.microsoft.com/ windows/downloads/bin/nts/DCOMtec.exe, 1996.

[CRW98] Microsoft Corporation, Redmond, and Washington. Microsoft Component Services: A Technology Overview. web: http://www.microsoft.com/com/wpaper/compsvcs.asp, 1998.

[CS96] Jin W. Chang and Colin T. Scott. Agent - based Workflow: TRP Support Environment (TSE). In *Fifth International World Wide Web Conference*, Paris, France, May 1996.

[CW98] Jonathan E. Cook and Alexander L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, July 1998.

[CY90] P. Coad and E. Yourdon. *Object-Oriented Analysis*. Yourdon Press, Prentice Hall, New Jersey, 1990.

[DBW98] Jean-Claude Derniame, Badara Ali Baba, and David Wastell. *Software Process: Principles, Methodology, and Technology*. Springer Verlag LNCS 1500, Berlin, Germany, 1998.

[DEA98] S. Dami, J. Estublier, and M. Amiour. APEL: A Graphical Yet Executable Formalism for Process Modeling. In *Process Technology edited by E. Nitto and Alfonso Fuggetta*, pages 61–96, Politecnico di Milano and CEFRIEL, 1998. Kluwer Academic Publishers.

[DeM82] Tom DeMarco. *Controlling Software Projects: Man-agement, Measurement and Estimation.* Yourdon Press Computing Series. Prentice Hall, Inc, 1982. ISBN 0-13-171711-1 025.

[Der92] Jean-Claude Derniame, editor. *Proc. Second European Workshop on Software Process Technology (EWSPT'92), Trondheim, Norway. 253 p.* Springer Verlag LNCS 635, September 1992.

[DFJN97] J. E. Doran, S. Franklin, N. R. Jennings, and T.J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[DGL86] Klaus Dittrich, Willi Gotthard, and Peter C. Lockemann. DAMOKLES — a Database System for Software Engineering Environments. In *[CDW87]*, pages 353–371, 1986.

[DLW00] Torgeir Dingsøyr, M. Letizia, and Alf Inge Wang. Teaching Software Process Improvement Through a Case Study. *Computer Applications in Engineering Education, Special Issue: Contributions from the International Conference on Engineering and Computer Education*, 8(3 and 4):229–234, 2000.

[Dor93] Alex Dorling. SPICE: Software Process Improvement and Capability dEtermination (ISO 15054). *Software Quality Journal*, 2:209–224, 1993.

[DT93] Flavio DePaoli and Francesco Tisato. Language Constructs for Cooperative Systems Design. In *[SP93]*, pages 329–343, 1993.

[Edw99] W.K. Edwards. *Core Jini*. Sun Microsystems Inc, Prentice-Hall Inc, 1999.

[Fei91] Peter H. Feiler. Configuration Management Models in Commercial Environments. Technical report, Carnegie-Mellon University, Software Engineering Institute, Pittsburgh, Pennsylvania, March 1991. 53 pp.

[Fel93] Stuart I. Feldman, editor. *Proceedings of the Fourth International Workshop on Software Configuration Management (SCM-4)*, Baltimore, Maryland, May 21–22, 1993.

[Fen91] Norman Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, 1991.

[Fer93] Christer Fernström. Process WEAVER: Adding Process Support to UNIX. In *[Ost93]*, pages 12–26, 1993.

[FFMM97] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an Agent Communication Language. In J. M. Bradshow et al., editors, *Software Agents*. MIT Press, 1997.

[FHA99] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addision Wesley, June 1999. Sun Microsystems Inc.

[FKN94] 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.

[FLM97] Tim Finin, Yannis Labrou, and James Mayfield. *Software Agents*, chapter KQML as an agent communication language. MIT Press, Cambridge, 1997. Ed. Jeff Bradshaw.

[Fug99] Hans Kristian Fuglenes. WWW technologies for handling process models. Technical report, Dept. of Computer and Information Science, NTNU, Norway, 1999.

[GB00] Jon Atle Gulla and Terje Brasethvik. On the Challenges of Business Modeling in Large-Scale Reengineering Projects. In Chen, Embley, Kouloumdjian, Liddle, and Roddick, editors, *Fourth International Conference on Requirements Engineering (ICRE'2000)*. Schaumburg, Illinois, June 2000.

[GHM96] Dimitrios Georgakopoulos, Mark F. Hornick, and Frank Manola. Customizing transaction models and mechanisms in a programmable environment supporting reliable workflow automation. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):630–649, August 1996.

[Gla99] Graham Glass. ObjectSpace, Overview of Voyager: ObjectSpace's Product Family for State-of-the-Art Distributed Computing. White paper, ObjectSpace, 1999. Availeble on web: http://www.objectspace.com/products /documentation/VoyagerOverview.pdf.

[Gra92] Robert B. Grady. *Practical software metrics for project management and process improvement*. Hewlett-Packard Professional Books. Prentice-Hall, 1992.

[Gra97] R. B. Grady, editor. *Successful Software Process Improvement*. Prentice Hall, 1997. ISBN 0-1362-6623-1.

[Gre98] Gregory Alan Bolcer. *Flexible and Customizable Workflow Execution on the WWW*. PhD thesis, University of California, Irvine, 1998.

[Gru94] Jonathan Grudin. Computer-Supported Cooperative Work: Its History and Participation. *IEEE Computer*, 27(5):19–26, 1994.

[GSSS92] Yaron Goldberg, Marilyn Safran, William Silverman, and Ehud Shapiro. Active Mail: A Framework for Integrated Groupware Applications. In D. Coleman, editor, *Groupware '92*, pages 222–224. Morgan Kaufmann Publishers, 1992.

[Håk94] Morten Håker. Evaluering av Prosjektstyringsverktøy, December 1994. 79 p. (diploma thesis). EPOS TR 239.

[Han00] Gisle Hannemyr. Cartoon Heroes Critical Reflections on Software Agents. In *Norsk Informatikkonferanse (NIK'2000)*, pages 79–90, Bodø, Norway, November 20-22 2000.

[HC$^+$88] Svein O. Hallsteinsen, Reidar Conradi, et al. The epos programming environment — an introduction. In *In [C$^+$88], p. 187–198*, 1988. (Also as DCST TR 58/88, EPOS TR 70, Jan 1989.).

[Hen88] Peter B. Henderson, editor. *Proc. 3rd ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments* (Boston)*, 257 p.,* November 1988. In ACM SIGPLAN Notices 24(2), Feb. 1989.

[Her97] Bjørn Hermans. Intelligent Software Agents on the Interne. *First Monday - Peer-reviewed Journal on the Internet*, 2(3), March 3rd 1997.

[Hew77] Carl Hewitt. Viewing Control Structures as Patterns of Passing Messages. *Journal of Artificial Intelligence*, 8(3):323–364, 1977.

[HL98] Arthur S. Hitomi and Dong Le. Endeavours and Component Reuse in Web-Driven Process Workflow. In *Proceedings of the California Software Symposium*, Irvine, California, USA, 23 October 1998.

[HM93] B. Heintz and M. Muller. *HOOD Reference Manual 3.1*. Masson, Paris, 1993.

[HMK+94] Volkmar Haase, Richard Messnarz, Günther Koch, Hans J. Kugler, and Paul Decrinis. BOOTSTRAP: Fine-Tuning Process Assessment. *IEEE Software*, pages 25–35, July 1994.

[HN86] A. Nico Habermann and David Notkin. Gandalf: Software Development Environments. *IEEE Transactions on Software Engineering (TSE)*, 12(12):1117–1127, December 1986.

[HN00] Anders Aas Hanssen and Bård Smidsrød Nymoen. DIAS II - Distributed Intelligent Agent System II. Technical report, Norwegian University of Science and Technology (NTNU), January 2000. Technical Report, Dept. of Computer and Information Science.

[Hol98] Steven Holzner. *XML Complete*. McGraw-Hill, 1998. ISBN 0-07-913702-4.

[Høy97] Geir Magne Høydalsvik. *Experiences in Software Process Modeling and Enactment*. PhD thesis, Department of Computer and Information Science, Norwegian University of Science and Technology, 9 March 1997.

[HP93] Hewlett-Packard. Developing SinverVision Processes. Technical Report Part Number: B3261-90003, Hewlett-Packard Company, 1993.

[Hum89] Watts S. Humphrey. *Managing The Software Process*. SEI Series in Software Engineering. 493 p. Addison–Wesley, 1989.

[IKV00] IKV. GrassHopper - The Agent Platform. web: http://www.ikv.de/products/grasshopper/, 2000.

[Ing98] Bent Ingebretsen. Erfaringsdatabase for firma X (in Norwegian), 2 March 1998. 206 p. + Confidential Appendix 14 p., EPOS TR 313 (diploma thesis).

[ISP00a] ISP. component - isp glossary definition and links. web: http://isp.webopedia.com/TERM/c/component.html, 2000.

[ISP00b] ISP. middleware - isp glossary definition and links. web: http://isp.webopedia.com/TERM/m/middleware.html, 2000.

[Jac96] M.Letizia Jaccheri. Reusing Software Process Models in E^3. In *The Tenth International Software Process Workshop*, 6, 1996.

[Jan95] Peter Janca. Pragmatic Application of Information Agents. In *BIS Strategic Decisions*, Norwell, United States, May 1995.

[JC93] M. Letizia Jaccheri and Reidar Conradi. Techniques for Process Model Evolution in EPOS. *IEEE Trans. on Software Engineering*, pages 1145–1156, December 1993. (special issue on Process Model Evolution).

[JGJ97] Ivar Jacobson, Martin Griss, and Patrick Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. ACM Press / Addison Wesley Longman, New York / Reading, Massachusetts, 1997.

[JHS$^+$99] 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.

[JL98] M. L. Jaccheri and P. Lago. How Project-based Courses face the Challenge of educating Software Engineers. In Jorge L. Diaz-Herrera, editor, *Proc. of the joint World Multiconference on Systemics, Cybernetics and Informatics (SCI'98) and the 4th International Conference on Information Systems Analysis and Synthesis (ISAS'98), Orlando, USA*, pages 377–385. Springer Verlag, LNCS 750, 1998. ISBN 980-07-5081-9.

[JLT99] Catholijn M. Jonker, Remco A. Lam, and Jan Treur. A Multi-Agent Architecture for an Intelligent Website in Insurance. In *Cooperative Information Agents III, Proceedings of the Third International Workshop on Cooperative Information Agents, CIA'99*, volume 1652 of *Lecture Notes in Artificial Intelligence*, pages 86–100, 1999.

[JPC00] Heecheol Jeon, Charles Petrie, and Mark R. Cutkosky. JATLite: A Java Agent Infrastructure with Message Routing. *IEEE Internet Computing*, 4(2), March/April 2000.

[JW99] M. L. Jaccheri and A.I. Wang. Software quality and software process improvement course home page, 1999. http://www.idi.ntnu.no/˜systprog (in Norwegian).

[Kai90a] Gail E. Kaiser. A flexible transaction model for software engineering. In *Proc. 6th International Conference on Data Engineering (ICDE '90),* pages 560–567, Los Angeles, CA, February 1990. IEEE Computer Society. Invited paper.

[Kai90b] Gail E. Kaiser. A flexible transaction model for software engineering. In *Proc. 6th International Conference on Data Engineering*, pages 560–567, Los Angeles, CA, February 1990. IEEE Computer Society. Invited paper.

[KJ95] R. Kehoe and A. Jarvis, editors. *Iso 9000-3: A Tool for Software Product and Process Improvement*. Springer, 1995. ISBN 0-387-94568-7.

[KK00] Lars Killingdal and Mufrid Krilic. Programmerbare transaksjoner – prosjektoppgave, April 2000. Student project report (pre-diploma), 133 p.

[KKB85] H. Korth, W. Kim, and F. Bancilhon. A Model of CAD Transactions. In *Proceedings of the 11th International Conference on Very Large Databases*, pages 25–33, 1985.

[Kon96] Jyrki Kontio. A Case Study in Applying a Systematic Method for COTS Selection. In H. Dieter Rombach, editor, *Proc. 18th Int'l Conf. on Software Engineering (ICSE'96), Berlin*, pages 201–209. ACM/IEEE-CS Press, N.Y., March 1996.

[KS97] G. Kotonya and I. Sommerville. Requirements engineering with viewpoints. In R.H. Thayer and M. Dorfman, editors, *Software Requirements Engineering*, pages 150–16, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[Kug86] Hans-Juergen Kugler, editor. *Information Processing'86*. North-Holland, IFIP, 1986.

[LA97] Danny B. Lange and Yariv Aridor. Agent Transfer Protocol – ATP/0.1. Technical report, IBM Tokyo Research Laboratory, March 19 1997. Availeble on web: http://www.trl.ibm.co.jp/aglets/atp/atp.htm.

[Lan97] Danny B. Lange. Java Aglet Application Programming Interface (J-AAPI) White Paper - Draft 2. Technical report, IBM Tokyo Research Laboratory, February 19 1997. Availeble on web: http://www.trl.ibm.co.jp/aglets/JAAPI-whitepaper.html.

[LB85] M. M. Lehman and L. A. Belady. *Program Evolution — Processes of Software Change*. Academic Press, 538 p., 1985.

[LC93] Chunnian Liu and Reidar Conradi. Automatic Replanning of Task Networks for Process Model Evolution in EPOS. In *[SP93]*, pages 434–450, 1993.

[LC98] Chunnian Liu and Reidar Conradi. Process View of CSCW. In Proc. of ISFST98, *Ocon Technology Application*, pages 46–51, Bremen, Germany, 15-17 September 1998. International Workshop on Intelligent Agents in Information and Process Management.

[Lea96] David Leare. *CBR – Experiences, Lessons and Future Directions*. AAAI/MIT Press, 1996.

[Leb94] David B. Leblang. The CM Challenge: Configuration Management that Works. In *[Tic94]*, chapter 1, pages 1–37. John Wiley, 1994.

[Leh69] Manny M. Lehman. The Programming Process. Technical Report Rep. RC 2722, IBM Research Centre, Yorktown Heights, NY 10594, September 1969.

[Leh87] M. M. Lehman. Process Models, Process Programming, Programming Support. In *Proc. 9th Int'l Conference on Software Engineering, Monterey, CA*, pages 14–16, March 1987. (Response to an ICSE'9 Keynote Address by Leon Osterweil).

[Leh94] Manny M. Lehman. Software Evolution. In *[Mar94]*, pages 1202–1208, 1994.

[Liu91] Chunnian Liu. An Expert System for Program and System Development. In *Proc. AVIGNON'91, Avignon, France, May 27–31, 1991, Volume 3*, pages 97–110, 1991.

[LMCL95] Jens-Otto Larsen, Bjørn P. Munch, Reidar Conradi, and Patricia Lago. Improving Cooperation Support in the EPOS CM System. In *Proc. 8th ERCIM Database Research Group Workshop on Database Issues and Infrastructure in Cooperative Information Systems, 23–25 Aug. 1995, NTH, Trondheim, Norway*, pages 135–147. ERCIM report 95-W002, SINTEF, 1995.

[LO98] Danny Lange and Mitsuru Oshima. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley, 1998.

[LS96] Mikael Lindvall and Kristian Sandahl. Practical Implications of Traceability. Technical report, Linköping University, 1996. To appear in Software Practice and Experience, 18 p.

[LST78] B. P. Lientz, E. B. Swanson, and G. Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6), June 1978.

[LvRH90] Ernst Lutz, Hans v.Kleist Retzow, and Karl Hoernig. MAFIA - An Active Mail-Filter-Agent for an Intelligent Document Processing Support. In S. Gibbs and A.A. Verrijn-Stuart, editors, *IFIP*, North-Holland, 1990. Elsevier Science Publishers B.V.

[Mad91] Nazim H. Madhavji. The process cycle. *Software Engineering Journal*, 6(5):234–242, September 1991.

[Mad92] Nazim H. Madhavji. Environment Evolution: The Prism Model of Changes. *IEEE Trans. on Software Engineering*, SE-18(5):380–392, May 1992.

[Mad95] R.J. Madachy. Knowledge-Based Risk Assessment and Cost Estimation. *Automated Software Engineering*, 2:219–230, 1995.

[Mar94] John J. Marciniak, editor. *Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

[MB95] M. M. Moore and T. Brennan. Process improvement in the classroom. In Rosalind L. Ibrahim, editor, *SEI Conference on Software Engineering Education*, pages 123–130. Springer Verlag, LNCS 895, 1995. ISBN 3-5405-8951-1.

[MCJOLW95] Bjørn P. Munch, Reidar Conradi, Minh Ngoc Nguyen Jens-Otto Larsen, and Per H. Westby. Integrated Product and Process Management in EPOS. *Journal of Integrated CAE, 30 p.*, 1995. (special issue on Integrated Product and Process Modeling).

[MDL87] Harlan D. Mills, M. Dyer, and R. Linger. Cleanroom Software Engineering. *IEEE Software*, 4(5):19–25, September 1987.

[MDP98] M. Matskin, M. Divitini, and S. Petersen. An Architecture for Multi-Agent Support in a Distributed Information Technology Application. In *International Workshop on Intelligent Agents in Information and Process Management*, page 12, Bremen, Germany, 15-17 September 1998.

[MG94] G. De Michelis and M.A. Grasso. Situating Conversations within the Language/Action Perspective: Teh Milan Conversation Model. In *CSCW'94*, Chapel Hill, North Carolina, USA, 1994.

[Mic99a] Sun Microsystems. Java Remote Method Invocation (RMI). web: http://www.sun.com/products/jdk/1.2/ docs/guide/rmi/, 1999.

[Mic99b] SUN Microsystems. JavaSpaces TM Specification. White paper, SUN Microsystems, January 25 1999. Availeble on web: http://www.sun.com/jini/specs/js.pdf.

[Mic99c] SUN Microsystems. Java(TM) IDL. web: http://java.sun.com/products/jdk/idl/, Updated 12 december 1999.

[Mic00] Sun Microsystems. Community resources (Jini Specification). web: http://www.sub.com/jini/specs, 2000.

[M.L98] M.L. Jaccheri and G.P. Picco and P. Lago. Eliciting Process Models in E3. *ACM Transactions on Software Engineering and Methodology*, 7(4):368–410, October 1998.

[MLG92] T. W. Malone, K.Y. Lai, and K.R. Grant. Agents for Infromation Sharing and Coordination: A History and Some Reflections. In J.M. Bradshaw, editor, *Software Agents,*, pages 109–143, Toronto, Canada, 1992. AAAI Press/The MIT Press.

[MLG$^+$93] Bjørn P. Munch, Jens-Otto Larsen, Bjørn Gulla, Reidar Conradi, and Even-André Karlsson. Uniform Versioning: The Change-Oriented Model. In *[Fel93]*, pages 188–196, 1993.

[MLL97] 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.

[MM97] Zakaria Maamar and Bernard Moulin. An agent-based approach for intelligent and cooperative systems. In *Knowledge and Data Engineering Exchange Workshop*, pages 19 –26. IEEE Computer Society Press, 1997.

[MPsP$^+$94] Frank McGarry, Rose Pajer-ski, Gerald Page, Sharon Waligora, Victor Basili, and Marvin Zelkowitz. Software Process Improvement in the NASA Software Engineering Laboratory. Technical report, NASA/Goddard Space Flight Center, Computer Sciences Corporation, University of Maryland, 1994. CMU/SEI-94-TR-22, ESC-TR-94-022.

[MSW90] N. Madhavji, W. Schaefer, and H. Weber, editors. *Proc. First International Conference on System Development Environments and Factories — SDEF'89, 9-11 May 1989, Berlin*, London, March 1990. Pitman Publishing, 241 p.

[Mul96] H. J. Muller. Negotiation Principles. In *Foundations of Distributed Artificial Intelligence*, pages 211–230. Wiley Interscience, 1996.

[Mun93] Bjørn P. Munch. *Versioning in a Software Engineering Database — the Change Oriented Way*. PhD thesis, DCST, NTH, Trondheim, Norway, August 1993. 265 p. (PhD thesis NTH 1993:78).

[NC94a] Minh Ngoc Nguyen and Reidar Conradi. Classification of Meta-processes and their Models. In *Proc. from the third International Conference on Software Process, Washington, USA, 10-11 October*, pages 167–175, 1994.

[NC94b] Minh Ngoc Nguyen and Reidar Conradi. Classification of Meta-processes and Their Models. In *[Per94]*, 1994. 167–175.

[NC96] Minh N. Nguyen and Reidar Conradi. Towards a Rigorous Approach for Managing Process Evolution. In *Carlo Montangero (Ed.): "Proc. 4th European Workshop on Software Process Technology (EWSPT'96)"*, pages 18–35, Nancy, France, 9–11 Oct. 1996. Springer Verlag LNCS 1149.

[Ngu97] Minh Ngoc Nguyen. *Framework and Approach for Managing Software Process Evolution in EPOS*. PhD thesis, IDI, NTNU, 22 May 1997. IDI-rapport 97:7, NTNU PhD thesis 1997:73, SU-report 6/97.

[NR69] Peter Naur and Brian Randell, editors. *Software Engineering – Proc. NATO Conference in* Garmisch-Partenkirchen*, 1968*. NATO Science Committee, Scientific Affairs Division, NATO, Brussels, January 1969.

[NWC97] Minh Ngoc Nguyen, Alf Inge Wang, and Reidar Conradi. Total Software Process Model Evolution in EPOS. In Richard N. Taylor and Alfonso Fuggetta (Eds.), editor, *Proc. 19th Int'l Conf. on Software Engineering (ICSE'97)*, pages 390–399. ACM/IEEE-CS Press, N.Y., May 1997.

[OB92] Markku Oivo and Victor R. Basili. Representing Software Engineering Models: The TAME Goal Oriented Approach. *IEEE Transactions on Software Engineering*, 18(10):886–898, October 1992.

[OKO98] Mitsuru Oshima, Guenter Karjoth, and Kouichi Ono. Aglets Specification 1.1 Draft. Technical report, IBM Tokyo Research Laboratory, September 8 1998. Availeble on web: http://www.trl.ibm.co.jp/aglets/spec11.html.

[OMG97] OMG. CORBA Components: Join Initial Submission. ftp: ftp://ftp.omg.org/pub/docs/orbos/97-11-24.pdf, 1997.

[Orl92] W.J. Orlikowski. Learning from Notes: Organizational Issues in Groupware Implementation. In *Proceedings of the Conference on Computer-Supported Cooperative Work, CSCW'92*, pages 362–369, Toronto, Canada, 1992. ACM Press.

[Ost87] Leon Osterweil. Software Processes are Software Too. In *Proc. 9th Int'l Conference on Software Engineering, Monterey, CA*, pages 2–13, March 1987. (Keynote address at the conference).

[Ost93] Leon Osterweil, editor. *Proc. 2nd Int'l Conference on Software Process (ICSP'2), Berlin. 170 p.* IEEE-CS Press, March 1993.

[OSVS82] T. W. Olle, H.G. Sol, and A. A. Verrijn-Stuart. Information Systems Design Methodologies: A Comparative Review. North-Holland, 1982.

[Øye98] Joar Øyen. Guidelines for Implementing Software Agent Architectures. Technical report, Norwegian University of Science and Technology (NTNU), December 1998. Technical Report, Dept. of Computer and Information Science, 161 p.

[Paw91] Z. Pawlak. *Rough Sets – Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, Boston, London, Dordrecht, 1991.

[PC94] Daniel J. Paulish and Anita D. Carleton. Case Studies of Software-Process-Improvement Measurement. *IEEE Computer*, pages 50–56, September 1994.

[PCCW93] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. The Capability Maturity Model for Software, Version 1.1. *IEEE Software*, pages 18–27, July 1993.

[Per94] Dewayne E. Perry, editor. *Proc. 3nd Int'l Conference on Software Process (ICSP'3), Washington, 187 p.* IEEE-CS Press, October 1994.

[PHBN99] 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.

[PMC96] ICL Enterprises Process Management Centre, Enterprise Technology. *ProcessWise Integrator, PML Reference*. Staffordshire, UK, first edition, April 1996.

[pro00] CAGIS project. Cooperative agents in global information space webpage. web: http://www.idi.ntnu.no/∼cagis, 2000.

[PSW92] B. Peuschel, W. Schaefer, and S. Wolf. A Knowledge-Based Software Development Environment (on MERLIN). *International Journal of Software Engineering and Knowledge Engineering*, 2(1):79–106, March 1992.

[PWCC95] Marc C. Paulk, Charles V. Weber, Bill Curtis, and Mary B. Chrissis. *The Capability Maturity Model for Software: Guidelines for Improving the Software Process*. SEI Series in Software Engineering. 640 p. Addison–Wesley, 1995.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, New Jersey, 1991.

[RBW00] Heri Ramampiaro, Terje Brasethvik, and Alf Inge Wang. Supporting Distributed Cooperative Work in CAGIS. In *4th IASTED International Conference on Software Engineering and Applications (SEA'2000)*, Las Vegas, Nevada, USA, 6-9 November 2000.

[Red97] Felix Redmill. *Software Projects – Evolutionary vs. Big-Bang Delivery*. John Wiley, New York, 1997. 254 p., ISBN 0 471 93343 0.

[RG96] Mark Roseman and Saul Greenberg. TeamRooms: Network Places for Collaboration. In M.S. Ackerman, editor, *CSCW'96 ACM Conferrence on Computer Supported Cooperative Work*, pages 325–333, Boston, MA, USA, 1996. ACM Press.

[RN00] Heri Ramampiaro and Mads Nygård. Cagistrans: A transaction framework to support cooperating agents. Technical Report IDI-5/00, Norwegian University of Science and Technology, 2000.

[Roc75] Mark J. Rochkind. The Source Code Control System. *IEEE Trans. on Software Engineering*, SE-1(4):364–370, 1975.

[Rol98] Rolf A. de By and Wolfgang Klas and Jari Veijalainen. *Transaction Management Support for Cooperative Applications*. Kluwer Academic Publishers, 1998.

[Roy70] W. W. Royce. Managing the Development of Large Software Systems: Concept and Techniques. In *Proceedings of WesCon*, pages 1–9, August 1970. Reprinted in Proc. Int'l Conf. Software Eng., IEEE Computer Society Press, 1987, pp. 328–338.

[Rr99] Heri Ramampiaro and Mads Nygård. Transaction support for cooperative environments: A state-of-the-art. Technical Report IDI-nr. 10/99, Norwegian University of Science and Technology, 1999.

[Sal01] Terje Salvesen. Global Agent Migration with DIAS III. Technical report, Norwegian University of Science and Technology (NTNU), January 29 2001. EPOS TR 408 (diploma project thesis), 98 p. + 45 p. App.

[Sar96] Sunil K. Sarin. Object-Orient Workflow Technology in InConcert. In *Forty-First IEEE Computer Society International Conference: Technologies for the Information Superhighway*, pages 446–450, Santa Clara, California, February 25-28 1996.

[SB92] Kjeld Schmidt and Liam Bannon. Taking CSCW seriously – supporting articulation work. *Computer Supported Work. An International Journal*, 1(1-2):7–40, 1992.

[SD97] Carla Simone and Monica Divitini. Ariadne: Supporting Coordination through a Flexible Use of the Knowledge on Work Process. *Journal UCS (Electronic Journal)*, 3(8), 1997. Also as NTNU/SU-report 23/97 and at http://www.iicm.edu/jucs_3_8.

[Sel00] Rune Selvåg. Web-transactions. Master's thesis, Norwegian University of Science and Technology, 2000.

[Sim98] Morten Simonsen. Diploma thesis: Renaissance Multimedia System. Technical report, Dept. of Computer and Information Science, NTNU, Norway, 1998.

[Som95] Ian Sommerville. *Software Engineering*. Addison-Wesley, 1995. ISBN 0-2014-2765-6.

[SP93] Ian Sommerville and Manfred Paul, editors. *Proc. 4th European Software Engineering Conference* (Garmisch-Partenkirchen, FRG)*, Springer Verlag* LNCS 717*, 516 p.*, September 1993.

[SS86] R.L. Schultz and D.P. Slevin. Project Implementation Profile (PIP). *Project Management Journal*, September 1986. Distributed in Compendium in course 92520 Project Organization, 1997 NTNU; translated to Norwegian by Telenor-Novit.

[STO99] J.W. Shepherdson, S.G. Thompson, and B.R. Odgers. Cross Organisational Workflow Co-ordinated by Software Agents. In *Workshop on Cross-Organisational Workflow Management and Co-ordination*, San Francisco, USA, February 1999.

[Sun89] Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043, USA. *The Network Software Environment – A Sun Technical Report*, 1989.

[SW00] Terje Salvesen and Jan Waage. DIAS III - Distributed Intelligent Agent System Using JavaSpaces. Technical report, Norwegian University of Science and Technology (NTNU), April 2000. EPOS TR 390 (pre-diploma project thesis), 118 p. + 21 p. App.

[SZBB86] Daniel C. Swinehart, Polle T. Zellweger, Richard J. Beach, and Robert B. Bagmann. A structured view of the Cedar programming environment. *ACM Transactions on Programming Languages and Systems*, 8(4):419–490, 1986.

[TBC$^+$88] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michael Young. Foundations for the Arcadia Environment Architecture. In *[Hen88]*, pages 1–13, November 1988.

[Ter99] Terje Brasethvik and John Atle Gulla. Semantically accessing documents using conceptual model descriptions. In P. Chen, D.W. Embley, and S.W. Little, editors, *Advances in conceptual modeling*, Paris, France, November 1999. WEBCM*99.

[Ter00] Terje Brasethvik and John Atle Gulla. Natural language analysis for semantic document modeling. In E. Metais, editor, *Proceedings on 5th International Conference on Application of Nature Language to Information Systems (NLDB'2000)*, Versailles, France, June 2000.

[TGML98] M.T. Tu, F. Griffel, M. Merz, and W. Lamersdorf. A Plug-in Architecture Providing Dynamic Negotiation Capabilities for Mobile Agents. In *2. Intl. Workshop on Mobile Agents (MA'98)*, Stuttgart, Germany, September 1998.

[Tia98] 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.

[Tic85] Walter F. Tichy. RCS — A System for Version Control. *Software — Practice and Experience*, 15(7):637–654, 1985.

[Tic94] Walter F. Tichy, editor. *Configuration Management*. (Trends in software). John Wiley, 1994. ISBN 0-471-94245-6.

[USK97] R. L. Upchurch and J. E. Sims-Knight. Designing Process-Based Software Curriculum. In *Proc. of the 10th ACM/IEEE-CS Conf. on Software Engineering Education and Training*, pages 28–38. IEEE Computer Society Press, April 1997. ISBN 0-8186-7886-0.

[Waa00] Jan Waage. COCAS - COnference Cooperative Agent System. Technical report, Norwegian University of Science and Technology (NTNU), December 2000.

[Wan95] Alf Inge Wang. Diploma thesis: Conflict HAndling Tool-kit eXtension. Technical report, Dept. of Computer Science, NTH, Norway, December 1995.

[Wan99] 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.

[Wan00a] Alf Inge Wang. Experience paper: Implementing a Multi-Agent Architecture for Cooperative Software Engineering. In *Twelfth International Conference on Software Engineering and Knowledge Engineering (SEKE'2000)*, Chicago, USA, 6-8 July 2000.

[Wan00b] Alf Inge Wang. Support for Mobile Software Processes in CAGIS. In Reidar Conradi, editor, *Seventh European Workshop on Software Process Technology*, Kaprun near Salzburg, Austria, 22-25 February 2000.

[Wan00c] Alf Inge Wang. Using Software Agents to Support Evolution of Distributed Workflow Models. In *Proc. International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000)*, page 7pp, Wollongong (near Sydney), Australia, December 12-15 2000.

[War89] Brian Warboys. The IPSE 2.5 Project: Process Modelling as the basis for a Support Environment. In *[MSW90], 26 p.*, May 1989.

[War94] Brian Warboys, editor. *Proc. Third European Workshop on Software Process Technology (EWSPT'94), Villard-de-Lans, France. 274 p.* Springer Verlag LNCS 772, February 1994.

[WBWW90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice Hall, New Jersey, 1990.

[WCL00] 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.

[Web00] Webopedia. Definitions and links: Software engineering. http://webopedia.internet.com/TERM/s/software_engineering.html, 2000.

[WF86] Terry Winograd and Fernando Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Addison-Wesley, 1986. ISBN: 0201112973.

[WfM99] WfMC. Workflow Management Coalition - Terminology & Glossary. Technical report, The Workflow Management Coalition, February 1999. Document Number WFMC-TC-1011, Availeble on web: http://www.aiim.org/wfmc/standards/docs/glossy3.pdf.

[WfM00] WfMC. Workflow Standard - Interoperability Wf-XML Binding. Technical report, The Workflow Management Coalition, May 1 2000. Document Number WFMC-TC-1023, Availeble on web: http://www.aiim.org/wfmc/standards/docs/Wf-XML-1.0.pdf.

[Whi96] Jim White. Mobile Agents White Paper. Technical report, General Magic, 1996. Availeble on web: http://www.ai.univie.ac.at/∼paolo/lva/vu-sa/html/white_whitepaper/.

[Whi97] Jim Whitehead. WebDAV: Evolving the Web into a read Write Medium. MSDN Online web: http://msdn.microsoft.com/workshop/standards/webdav.asp, April 7 1997.

[Wil91] P. Wilson. *Computer Supported Cooperative Work: An Introduction*. Oxford, Intellect Books, 1991.

[WJ95] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[WLC99] 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.

[WLCM98a] Alf Inge Wang, Jens-Otto Larsen, Reidar Conradi, and Bjørn Munch. Improving Cooperation Support in the EPOS CM System. In Volker Gruhn, editor, *Proc. EWSPT'98, Weybridge (London), 18-19. Sept. 1998, Springer Verlag LNCS 1487*, pages 75–91, September 1998.

[WLCM98b] Alf Inge Wang, Jens-Otto Larsen, Reidar Conradi, and Bjørn Munch. Improving Cooperation Support in the EPOS CM System. In Volker Gruhn, editor, *Proc. EWSPT'98, London, 18-19. Sept. 1998*, page 17, September 1998.

[WRH$^+$00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wésslen. *Experimenation in Software Engineering - An Introduction*. Kluwer Academic Publishers, Boston / Dorrecht / London, 2000.

[WW98] E. James Whitehead and Meredith Wiggins. WEBDAV: IETF Standard for Collaborative Authoring on the Web (Web-based Distributed Authoring and Versioning). *IEEE Internet Computing*, pages 34–40, Sept./Oct. 1998.

[XML99] XML.COM. XML.COM - XML Implementations. web: http://www.xml.com/xml/pub/Guide/XML_Implementations, 1999. (C) Seybold Publications and O'Reilly and Associates, Inc.

[Yar98] Yariv Aridor and Danny B. Lange. Agent Design Patterns: Elements of Agent Application Design. In Katia P. Sycara and Michael Wooldridge, editors, *Proc. Second International Conference on Autonomous Agents*, pages 108–115, St. Paul, Minnepolis, USA, May 9-13 1998. ACM Press, New York.

[YC75] E. Yourdon and L. L. Constantine. *Structured Design*. Yourdon, Inc., New York, 1975.

[Yeo96] Benjamin Yeomans. Enhancing the world wide web. Technical report, Computer Science Dept., University of Manchester, 1996. Supervisor: Prof. Brian Warboys.

[YL99] 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.

[You94] Patrick S. Young. *The Teamware Language Reference Manual*. Teamware, March 1994.

[Zah98] S. Zahran, editor. *Software Process Improvement*. Addison Wesley, 1998. ISBN 0-2011-7781-X.

[ZW98] Marvin V. Zelkowitz and Dolores R. Wallace. Experimental Models for Validating Technology. *IEEE Computer*, 31(5):23–31, May 1998.