

Using software agents to support evolution of distributed workflow models

Alf Inge Wang

Norwegian University of Science and Technology (NTNU)

N-7491 Trondheim, Norway

Email: alfw@idi.ntnu.no, Phone: +47 73594485

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 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 instantiated. 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*, which can individually be changed locally. This gives users the opportunity for local adoptions, and therefore makes 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 an activity, re-allocating activities, and re-scheduling activities.

In [4], 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: **(1) Customer postponement** 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 [11], 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 are 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 often affect 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 has been focusing on database and transaction support for these systems. In [2] 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 [7] 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 in stead of exchange of data as in Co.

2 The CAGIS Process Centred Environment

In 1997, a project called *Cooperative Agents in Global Information Space* (CAGIS) [5] was started. A main goal 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 instantiated 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 be changed 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 [8, 9].

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 specifies 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, which 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 act 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 proactiveness. 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 (processing tools and others). It can be private, and 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, which 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 [12, 6, 3].

- **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 [10, 1].

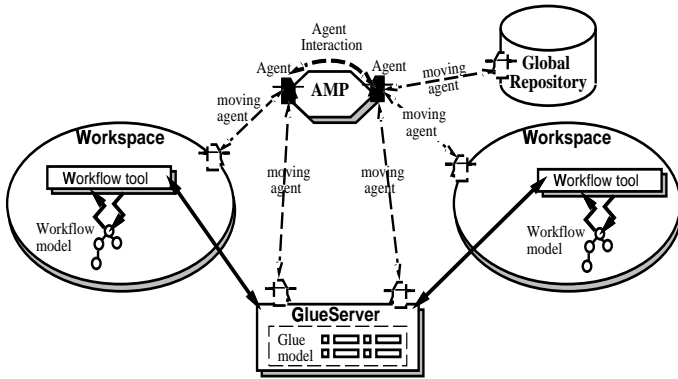


Figure 1. The CAGIS Process Centred Environment

Figure 1 shows a simplified illustration of how the different components in the CAGIS PCE interact. In Figure 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).

3 The Problem scenarios

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.

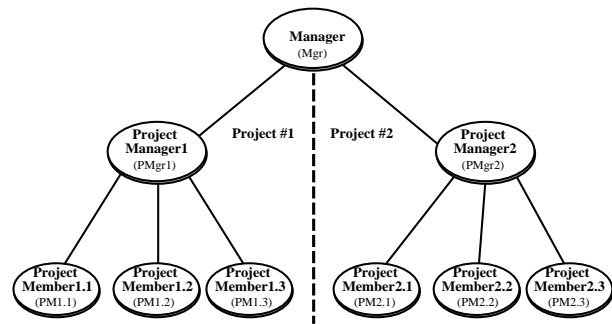


Figure 2. Organisational hierarchy used in the scenarios

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 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.

Note that the shape of the organisation is created to make it easier to illustrate the different scenarios.

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 frag-

ment **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 (s)he 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 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 many 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.

3.2 Scenario 2: Exclusive update on one process fragment

As for scenario in section 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* wants 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?

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 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 inter-relationships, only the project

managers and the *Manager* can make changes. Typical dependencies between process fragments are here pre-order relationships between activities.

Here problems can occur 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?

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* recognises 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 many 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.

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 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 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 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 must at least have as strict access policy as its parent workspace. We can identify four levels of access policies according to strictness (1 is most strict, 4 is least strict):

1. Exclusive access (section 3.4)
2. Exclusive update on related process fragments (section 3.3)
3. Exclusive update on one process fragment (section 3.2)
4. Anarchy (section 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. 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.

4 Awareness Agents dealing with Workflow Evolution

From the scenarios in section 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 3, an overall design of the workspace manager is illustrated.

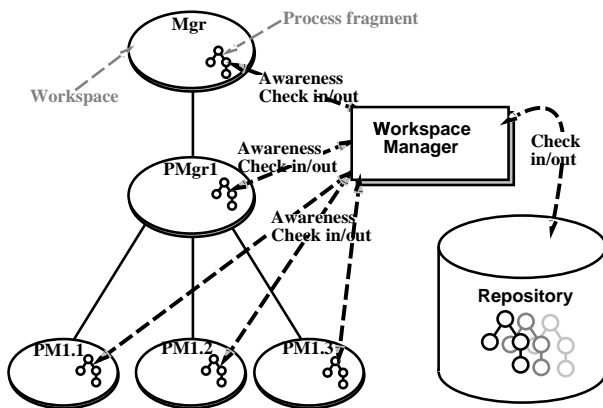


Figure 3. Illustration of the workspace manager

The figure shows how the workspace manager interacts 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 workspaces, managed by the workspace

manager. In addition, we need awareness support to deal with consistency problems. We suggest using 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 3 is supported by the workspace manager and software agents.

4.1 Support for scenario 1

In scenario 1, there are 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 **Solo-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 **Solo-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 a negotiation about who is going to get the access to update the process fragment **Solo-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*, which 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 the process fragment is checked back to the repository through the workspace manager, changes must 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 changes made to a process fragment in one workspace, will be visible in the other and vice versa).

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 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.

4.3 Support for scenario3

Here we have to solve the problem on 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 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.

4.4 Support for scenario4

When there is only one person having access (both read and write) to a process fragment at time, 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.

4.5 Support for scenario5

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 at least have as strict access policy as its parent workspace according to the four levels of strictness as listed in section 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*, which 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 have 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 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.

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) used by agents: *ask-if*, *tell*, *un-tell*, *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, communicate 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.

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 extensively 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, and Heri Ramampiaro for giving me useful advices for this paper.

References

- [1] 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.
- [2] Claude Godart. Coo: A transaction model to support cooperation software developers coordination. In *4th European Software Engineering Conference*, pages 361–379, Garmisch, Germany, 1993. LNCS 717.
- [3] Anders Aas Hanssen and Bård Smidsrød Nymoén. 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.
- [4] Minh Nguyen, Alf Inge Wang, and Reidar Conradi. Total Software Process Model Evolution in EPOS. In *Proceedings ICSE'97*, Boston, USA, May 1997.
- [5] CAGIS project. Cooperative agents in global information space webpage. web: <http://www.idi.ntnu.no/~cagis>, July 1997.
- [6] Geir Prestegård, Anders Aas Hanssen, Snorre Brandstadmoen, and Bård Smidsrød Nymoén. DIAS - Distributed Intelligent Agent System. Technical report, Norwegian University of Science and Technology (NTNU), April 1999. Technical Report, Dept. of Computer and Information Science, 387 p.
- [7] Rolf A. de By and Wolfgang Klas and Jari Veijalainen. *Transaction Management Support for Cooperative Applications*. Kluwer Academic Publishers, 1998.
- [8] 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.
- [9] Alf Inge Wang. Support for Mobile Software Processes in CAGIS. In *Seventh European Workshop on Software Process Technology*, Kaprun near Salzburg, Austria, 22-25 February 2000.
- [10] Alf Inge Wang, Reidar Conradi, and Chunnian Liu. Integrating Workflow with Interacting Agents to support Cooperative Software Engineering. Submitted to Software Engineering and Applications' 2000 (SEA'2000).
- [11] 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.
- [12] Alf Inge Wang, Chunnian Liu, and Reidar Conradi. A Multi-Agent Architecture for Cooperative Software Engineering. In *Proc. of The Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, pages 1–22, Kaiserslautern, Germany, 17-19 June 1999.