

Improving Cooperation Support in the EPOS CM System

Alf Inge Wang, Jens-Otto Larsen, Reidar Conradi and Bjørn P. Munch.

Norwegian University of Science and Technology (NTNU)
N-7034 Trondheim, Norway (alfw@idi.ntnu.no)

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 conflicts¹ 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.

Keywords: Configuration management, awareness, planning, conflict solving, cooperative support, CSCW, consistency, process modeling.

2 INTRODUCTION

One of the problems in software configuration management regards cooperative work on large systems, where several people are doing development simultaneously, and their 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other hand, (s)he may get in trouble when the work is finished, and turns out to be “wrong” because it is based on an outdated version of the same library/header file.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

¹ Access conflict does in this paper mean that two or more people want to change the same file at the same time.

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 3 discusses support for cooperative work within some existing CM and similar systems. Section 4 discusses about what cooperative support a CM-system should have and presents a list of cooperative support requirements for the EPOS-CM system. Section 5 is a short description of the EPOS CM system and what cooperative support we have added to the system. Section 6 describes the cooperative support we have added in EPOS. Section 7 presents some experiences we have had using the cooperative support in ECM. Section 8 discusses how cooperative CM support is related to the process modeling domain. The last section concludes this paper.

3 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.

3.1 SCCS/RCS

SCCS [18] and RCS [23] 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.

3.2 ClearCase

ClearCase [11] 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.

3.3 NSE

NSE [21, 5] (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.

3.4 Adele

Adele [1], 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.

3.5 Lotus Notes

Lotus Notes [15] 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.

3.6 Other Systems

Some dedicated Software Engineering DBMSes, like DAMOKLES [4], provide open and flexible long transaction models, but lack a user framework that enables cooperation support. Software engineering environments like Marvel [8], COO [7], and our own EPOS [14] all provide some cooperation functionality based on their own databases.

More traditional groupware systems like TeamRoom [19], BSCW [17] and Object Lens [22] 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.

4 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* [9], or workspaces in the CM context.

From the discussion in section 3, 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. Examples of useful workspace information can be: Who is connected to a workspace, what files are changed and by whom, what files are read within a workspace, when was a file changed etc.
3. *Awareness*: Providing support for notifying users about events that will affect their work in their respective workspaces. Users should be warned if two or more users want to change the same file at the same time and they should also be warned if an updated version of a file they work on exists. This service will help users to know who they must negotiate and cooperate with.
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*²: 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.
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). This is necessary functionality to support cooperative processes, e.g., copying an updated component to a workspace to get the most recent version of the component, copy component between workspaces to merge changes made to the component etc.

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

5 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 [10].

Figure 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 [12]. The ECM tool has both a graphical and command-line user interface and provide support for:

² 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).

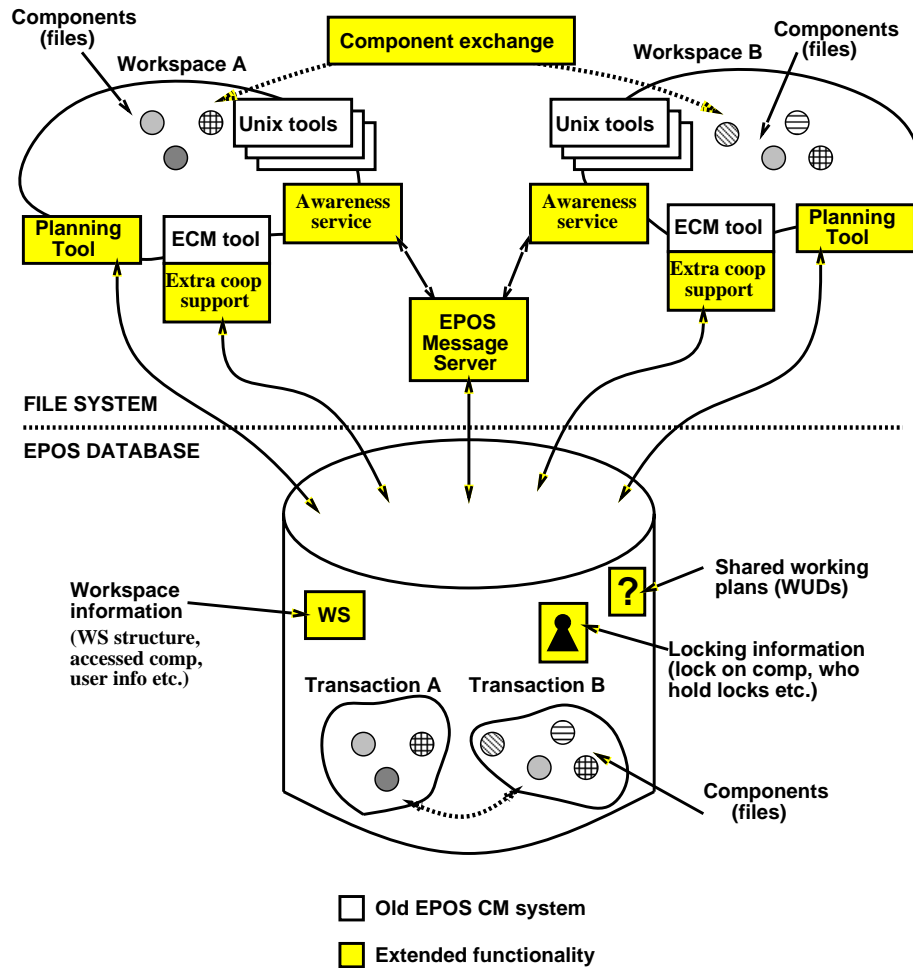


Fig. 1. EPOS CM with cooperative support added.

- **Manage workspaces**³: Navigate in workspace hierarchy³ and create new/ abort/commit workspaces.
- **Manage components**⁴: Navigate in the product space, check in and out components etc.

The EPOS Database [12] is a general-purpose DBMS with features geared for CM. All objects are versioned according to our COV versioning model [13],

³ EPOS supports nested workspaces which is represented in the repository as nested transactions

⁴ The term component in EPOS is used to name products, files etc.

which we will not discuss in this paper.

The shaded blocks in figure 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
- **Planning tool:** Plan file access in advance to avoid conflicts.

More detailed description of these features is shown in section 6

6 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.

6.1 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 configuration⁵ 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 2.

⁵ The version configuration is in EPOS represented through ambition and choice as described in [12].

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.

A WUD contains the following pieces of information about what will take place during the execution of a transaction:

- Properties of the workspace and transaction: status (planned, running, committed, aborted), owner/responsible, planned start and duration, whether it is decomposable, etc.
- Hierarchy, i.e. name of parent transaction, if any.
- Versioning context.
- Group name of the products to work on.
- Initial read-set and write-set of transaction.
- Access-policies: lock and access modes.
- Cooperation policies and automatic actions in case of conflicting access.
- General work-mode: configuring how ECM deals with conflicting component access. Currently we use four modes: `serial`, `coop_immediate`, `coop_commit`, or `coop_deferred`. The first will guarantee that no conflicts occur, while the other ones differ in when the conflicts must be resolved: immediately, before commit, or at a later stage (using versioning to separate concurrent updates).

Work-units are written in a text-editor and then parsed and stored in the database by using a WUD install tool. ECM workspaces can be started based on stored work-unit descriptions, and we plan to provide an extended interface to ECM which follow the policies and behavior specified in a WUD. The stored WUDs can be used for longer-term planning of work.

We have implemented a planning tool [3], as shown in figure 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.

WUDL is 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.

6.2 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.

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

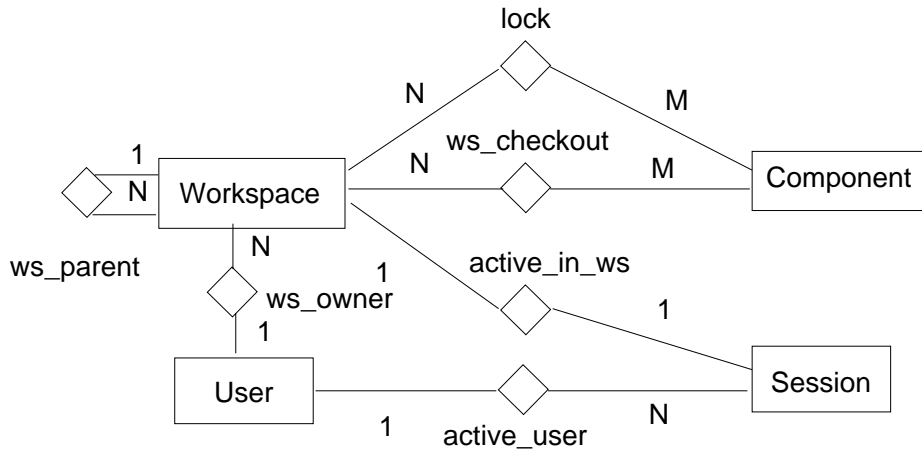


Fig. 2. Schema for transaction and workspace database.

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 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 6.1. The **lock** relation type models locked components, and similarly the **ws_checkout** 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 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.

6.3 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 6.4.

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. The negotiation tool support is based on the ECM Message Services (see section 6.4).
- **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. This means that all the changes done in that particular workspace 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. This tool is based on the EPOS Message Services and has built in support for copying components between workspaces and merging them. The synchronization tool should be used in situations where a common header/library file is edited in more than one workspace.

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. The user have to approve the merged version of the components. You are also allowed to specify your own merge tool.

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. The user has three options of how (s)he wants the update service to work.

1. **Get a notification only** The user receive a notification if more recent version of a component you have checked out exists. The notification can be browsed using the message browser provided in the ECM Tool (as described in next section).
2. **Ask to update** If a newer version of a component you have checked out exists, you will be asked if you want to replace the component.
3. **Automatically updated** If updated version of a component you have checked out exists, this component will automatically replace the old one. If the component is checked out for updating, you will have to merge the changes of the new version.

More detailed information about the awareness support in EPOS is described in [25].

6.4 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.

6.5 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 1).

These lock-modes are:

- *No locks*
- *Non-restrictive read locks* that permits other workspaces to hold write locks on the same objects.
- *Restrictive read locks* that prevent other workspaces from placing write locks on an object.
- *Exclusive read locks* that only allows less restrictive read locks by other workspaces (typically used to signal that the read lock will probably be upgraded to a write lock).
- *Multiple write locks* that is used for allowing several workspaces to update an object simultaneously (and merge changes later).
- *Non-restrictive write locks* that allow other workspaces to place read-locks on the same objects.
- *Exclusive write locks* that prevent any other access locks.

The default lock mode used in the workspace is one of the parameters given when starting a workspace, but lock modes can be specified individually for each component checked out. Workspaces are terminated with either an `ENDWORKSPACE` or an `ABORTWORKSPACE` command.

6.6 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.

7 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 all-though 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 seems 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.

8 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 seems 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 4 and 6, 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 6.2 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.

A possible extension to the cooperative support in ECM is trigger-mechanism (like in Adele). Triggers can be seen as light-weight process support and is often

denoted as process rules. In practice, this means that it is possible to specify what actions should be fired when a notification is received (e.g., use the component exchange feature in ECM to copy a component from another workspace when finished updating the component notification is received).

In most cases, traditional process models can not provide any support for cooperative processes. This means that researchers should look at what active support should be provided for cooperative processes. Such active support must provide little overhead work, be simple and flexible.

9 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.

References

1. Nouredine Belkhatir, Jacky Estublier, and Walcelio Melo. Software Process Model and Work Space Control in the Adele System. In [16], pages 2–11, 1993.
2. 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.
3. Reidar Conradi, Chunnian Liu, and Marianne Hagaseth. Planning Support for Cooperating Transactions in EPOS. *Information Systems*, 20(4):317–326, June 1995.

4. Klaus Dittrich, Willi Gotthard, and Peter C. Lockemann. DAMOKLES — a Database System for Software Engineering Environments. In [2], pages 353–371, 1986.
5. Peter H. Feiler. Configuration Management Models in Commercial Environments. Technical report, Carnegie-Mellon University, Software Engineering Institute, Pittsburgh, Pennsylvania, March 1991. 53 pp.
6. Stuart I. Feldman, editor. *Proceedings of the Fourth International Workshop on Software Configuration Management (SCM-4)*, Baltimore, Maryland, May 21–22, 1993.
7. Claude Godart. COO: A transaction model to support COOperating software developers COOrdination. In [20], pages 361–379, 1993.
8. 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.
9. 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.
10. 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.
11. David B. Leblang. The CM Challenge: Configuration Management that Works. In [24], chapter 1, pages 1–37. John Wiley, 1994.
12. 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).
13. Bjørn P. Munch, Jens-Otto Larsen, Bjørn Gulla, Reidar Conradi, and Even-André Karlsson. Uniform Versioning: The Change-Oriented Model. In [6], pages 188–196, 1993.
14. Bjørn P. Munch, Reidar Conradi, Jens-Otto Larsen, Minh Ngoc Nguyen, and Per Harald Westby. Integrated Product and Process Management in EPOS. *Journal of Integrated CAE*, 1995. (Forthcoming in special issue on Integrated Product and Process Modeling), 30 p.
15. 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. The Association for Computer Machinery, ACM Press.
16. Leon Osterweil, editor. *Proc. 2nd Int'l Conference on Software Process (ICSP'2), Berlin. 170 p.* IEEE-CS Press, March 1993.
17. T. Horstman R. Bentley and J. 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.
18. Mark J. Rochkind. The Source Code Control System. *IEEE Trans. on Software Engineering*, SE-1(4):364–370, 1975.
19. Mark Roseman and Saul Greenberg. TeamRooms: Network Places for Collaboration. In M.S. Ackerman, editor, *CSCW'96 ACM Conference on Computer Supported Cooperative Work*, pages 325–333, Boston, MA, USA, 1996. The Association for Computer Machinery, ACM Press.

20. Ian Sommerville and Manfred Paul, editors. *Proc. 4th European Software Engineering Conference* (Garmisch-Partenkirchen, FRG), Springer Verlag LNCS 717, September 1993.
21. Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043, USA. *The Network Software Environment – A Sun Technical Report*, 1989.
22. K.Y. Lai T. W. Malone and K.R. Grant. Agents for Information 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.
23. Walter F. Tichy. RCS — A System for Version Control. *Software — Practice and Experience*, 15(7):637–654, 1985.
24. Walter F. Tichy, editor. *Configuration Management*. (Trends in software). John Wiley, 1994. ISBN 0-471-94245-6.
25. Alf Inge Wang. Diploma thesis: Conflict handling tool-kit extension. Technical report, IDT, NTH, 1995.