A model for Semi-(a)Synchronous Collaborative Editing

Sten Minör & Boris Magnusson

LU-CS-TR:93-109

Also published in Proceedings of Third European Conference on Computer-Supported Cooperative Work - ECSCW'93, Milano, Italy, Kluwer Academic Publishers, 1993

Department of Computer Science
Lund University

P.O. Box 118
S-221 00 Lund
Sweden

A Model for Semi-(a)Synchronous Collaborative Editing

Sten Minör & Boris Magnusson
Department of Computer Science, Lund University, Sweden

Abstract: This paper presents a new model for semi-synchronous collaborative editing. It fills the gap between asynchronous and synchronous editing styles. The model is based on hierarchically partitioned documents, fine-grained version control, and a mechanism called active diffs for supplying collaboration awareness. The aim of the model is to provide an editing style that better suits the way people actually are working when editing a document or program together, using different writing strategies during different activities.

1. Introduction

During the last couple of years, a number of collaborative editors have been developed. The aim of the different systems vary, some are specifically supporting collaborative authoring, some are general purpose text editors, some support collaborative sketching or drawing, and some provide a framework for integrating existing editors into a collaborative environment. Furthermore, different systems are based on different architectures, provide sharing at different levels, and use different strategies for distribution. Despite the different goals and architectures the systems support two main editing styles: synchronous and asynchronous editing.

1.1 Synchronous editors

Up till now, work on synchronous editing has been dominating within the area of collaborative editing. A synchronous editor allows multiple users to access and edit a document (a text, picture, drawing, etc.) simultaneously. Since simultaneous editing operations performed on shared material by different users may conflict, most editors have a protocol to ensure consistency. This may either be done by some locking mechanism where a user explicitly locks an object before editing it to avoid conflicts, implicit locking where the system locks an object edited by a user, or by ordering editing events ensuring consistent updating at the different sites. Some synchronous editors support shared views and telepointers. A shared view allows different users to see a part of a document in exactly the same manner using the WYSIWIS (What You See Is What I See) metaphor. The user interfaces are here tightly coupled and if one user scrolls a window, for instance, the same window will be scrolled at the other sites. Telepointers allow multiple cursors, one for each user, which are shown at all sites and are updated in real time. Furthermore, some systems provide support for "meta work", i.e. communication about the work the system primarily supports, e.g. in the form of shared workspaces for text and drawings and support for speech and image communication. Examples of different kinds of collaborative editors supporting synchronous editing are: GroupSketch (Greenberg and Bohnet, 1991), a shared workspace for sketching, GROVE (Ellis, Gibbs, and Rein, 1991), a textual multi-user outlining tool, ShrEdit (McGuffin and Olson, 1992), a multi-user text editor, and DistEdit (Knister and Prakash, 1990), a toolkit for implementing distributed group editors.

1.2 Asynchronous editors

Conventional text editors and drawing tools are typically asynchronous. Even though they may run in a distributed environment with a shared file system they do not support collaboration by multiple users. Prep (Neuwirth, Kaufer, Chandhok, and Morris, 1990) is an example of an asynchronous editor supporting collaboration. It allows only one user to edit a document at a time, but has specialized support for commenting a document. A document is organized as a number of columns. The author may create the document contents in one column and a reviewer, for instance, may create a new column for his/her comments and bind the different comments to places in the author's column. The Prep editor thus mainly supports asynchronous collaboration for authoring in form of reviewing and commenting.

1.3 Synchronous or asynchronous editing?

An interesting question is if the separation into synchronous and asynchronous editing is for good. Does an editor of one of these categories really meet the users' needs in different tasks and different situations? We believe not.

Asynchronous editors may be useful for some tasks, e.g. when one person is commenting the work of another. This is intrinsically an asynchronous task where the reviewer typically finishes the work and returns it to the author. However, asynchronous editors do not allow people to work simultaneously on the same document. If this shall be achieved the document has to be partitioned and later joined manually by the users. While working on their own fragment, the users are not aware of what other users are doing meanwhile. Another option is to serialize the work, which, of course, is not desirable.

Synchronous editors support sharing of a document and awareness about other users' work, but they do not support the asynchronous working style. Synchronous editing may be very useful in the brainstorming phase of authoring a paper or in initial design of a program. It may also be useful when discussing the contents of the document with other users. However, it assumes the collaborating users to be *present*. If, for instance, two persons are co-authoring a paper, they do not work simultaneously all the time. One person may be away for some time (an hour, a day, a week). When the work is continued by that person he/she is not primarily interested in what the other person is doing right now, but rather what has happened in the document since the last time.

The fact that different situations demand different editing styles has been acknowledged before. Dourish and Belotti (1992) calls for an editing model which supports both synchronous and asynchronous work and a smooth transition between them. The SEPIA hypertext authoring system (Haake and Wilson, 1992) allows work in modes called: *individual*, *loosely coupled*, and *tightly coupled* respectively and switching between them. The *individual* mode works like traditional asynchronous editing. In *loosely coupled* mode a node may be edited by one user at a time while other users may see the changes, i.e. synchronous editing based on locking. Finally, *tightly coupled* mode adds shared views, telepointers, and audio communication.

An empirical study of how people actually are collaborating in writing is presented by Posner and Baecker (1992). A number of different writing strategies used in different phases of the authoring of a document are identified. They conclude that both synchronous and asynchronous strategies are used in different phases of a collaborative writing project and that a system must support both styles and a smooth transition between them. An interesting result of the study is that most collaborative writing projects used the "separate writers strategy", i.e. an asynchronous style of authoring, extensively.

In our own experience from software development, several of the observations by Posner and Baecker are valid also for explorative software development. A synchronous style of work is often used in the initial phases of development, e.g. brainstorming and initial design. For more detailed design and implementation the work is often split up and the work is mostly done asynchronously on a separate fragment of the design. When it comes to integration, testing, and debugging the work style turns to be more synchronous again. The situation seems to be similar to authoring. Collaborative software environments thus have to support both synchronous and asynchronous working strategies and smooth transitions between them.

In the next section we present a model which supports semi-synchronous editing. It is a general editing model which may be used as a basis both for authoring and software development. In section 3 the properties of the model is discussed. In section 4 some notes on the implementation and future work is given followed by conclusions in section 5.

2. A semi-synchronous editing model

The editing model presented here has been developed as a part of an ongoing project on collaborative software development environments. It is based on previous work in the Mjølner project, a project on object-oriented software development (Knudsen, Löfgren, Madsen, and Magnusson (1993), (Magnusson, Minör, and Hedin, 1990). However, we believe the basic editing model is more widely applicable. Thus we present it in more general terms. In our view the editing model is a general technique for semi-synchronous editing which may be the basis for application specific environments, e.g. authoring systems or software development environments.

First we present the notion of hierarchical documents and hierarchical browsing which is fundamental in our model. We then describe the fine-grained version control functionality, which automatically keeps track of the modifications of a document and supports simultaneous editing by different users and merging. Finally, we present active diffs which are used for continuously making the users aware of changes done by other users thus giving the editor a synchronous appearance.

2.1 Hierarchical documents

A document is organized as a hierarchical structure. The hierarchy corresponds to chapters, sections and paragraphs in a book or to blocks, classes, and procedures in programming languages. This hierarchy is fundamental in our model. A document is displayed using this hierarchy, it is edited in terms of the hierarchy, all elements in the hierarchy are version controlled, and the database server stores documents in terms of the hierarchy.

Figure 1 shows an example of the users view of a document. In the figure there are two main objects, the hierarchical document at the top and an evolution graph below, explained later. The document ”ECSCW’93” contains a number of parts: the title, an abstract, section 1-4, and a references part. All these parts are shown and manipulated by a hierarchical browser, which allows parts at one level to overlap but not to be moved outside its parent window. Subparts are added and removed in the browser by selecting entries from a menu, e.g. “new Section”. The parts in the figure contain text which is edited by a text editor.

The subdivision into particular parts is not built into the system but is described in a grammar specifying the hierarchy. The example in figure 1 shows a rather flat hierarchy. Using another grammar, the sections may contain subsections, which in turn may contain paragraphs and sub-paragraphs, etc. This is a matter of tailoring the system, which can be different for different groups or projects. Furthermore, by

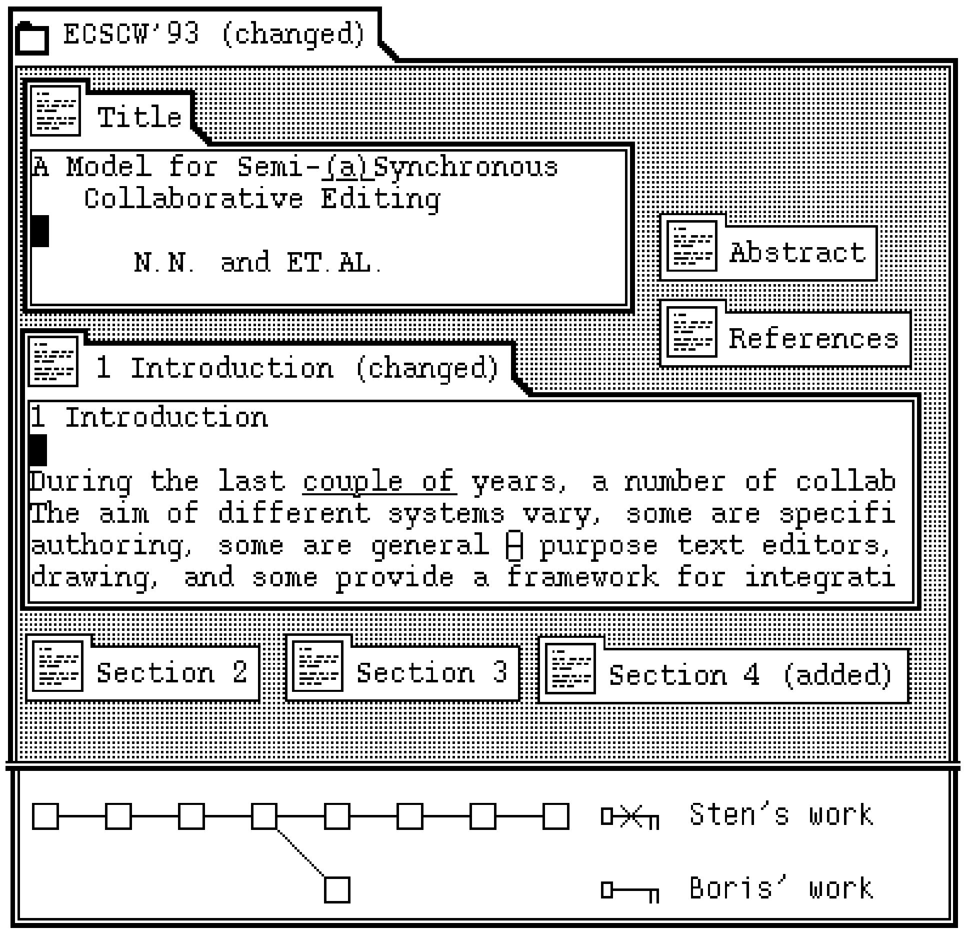

Figure 1 A hierarchical document

supplying different grammars, the browser may be tailored for different application areas, e.g. software development where the sub-parts may consist of classes, modules, procedures, functions, and methods, as long as the document forms a hierarchy.

The approach for hierarchical browsing initially was developed in the Mjølner Orm software environment (Hedin and Magnusson, 1988) where it turned out to be a useful and "intuitive" way of viewing and editing program structures. The explicit view of the logical structure of a document can in a collaborative environment be used as a means for organizing the work, where the users share the same partitioning of the document.

2.2 Fine-grained version control

Our editing model heavily relies on version control. In software engineering, version control is an established technique for dealing with complex systems developed by a team of system developers (Tichy, 1988), (Gustavsson, 1990). In our view, version control is a crucial technique for all collaborative design environments and editors. In such systems we do not only have to deal with potentially complex designs or documents but also with modifications of a document performed by a possibly large group of users simultaneously or at different times. Version control is important for several purposes. One is to keep track of the evolution history of a document. Another is to maintain consistent configurations of a complex design. It may also be used as a means for coordinating activities between several project members. All these aspects are important for collaborative systems.

In our editing model, hierarchical documents are version controlled. The lower part of figure 1 shows an evolution graph of the document “ECSCW’93”. A graph contains three different evolution steps: versions, alternatives and alternative merging as depicted in figure 2.

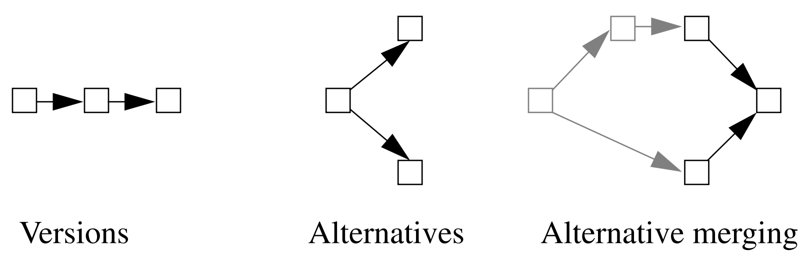

Figure 2 Evolution steps in an evolution graph

A version[1] is one step in the evolution of a document. It may contain arbitrary changes compared to its predecessor. Typically it may contain new chapters or paragraphs, parts may have been removed, or it may contain minor modifications such as corrections of spelling errors. A version is never changed once it has been established. The only way of modifying a document is to create a new version (or alternative) with the desired modifications.

An alternative (sometimes called variant) serves two main purposes. One is to split up the evolution into two (similar) documents with different purposes. A document may, for instance, be modified into two alternatives for two groups of readers, novices and experts, but with a core of common contents. The other purpose is to support simultaneous development among multiple users. If one user wants to edit a version that is edited by another user, an alternative is created instead of a version. The users edit their own alternatives and merge them into one version when ready.

Merge is taking place between alternatives which always have a common root, one or several steps away as shown in figure 2. All the concatenated changes in the alternatives (starting from the root version) are candidates for inclusion in the

1. Usually a distinction is made between versions and revisions. We do not do that distinction here, but use the term *version* in a generic fashion.

merged version. If there are conflicting changes the decision on what to include has to be made by the person doing the merge who can also do any other changes to the document at the same time. Conflicts might occur on the lexical, syntactical or semantic levels and automatic detection of such conflicts can be perceived at least at the primitive level. This schema can be used when merging two or more alternatives although it might be practical to merge two alternatives at the time.

In our model, all subparts of a hierarchical document are subjected to version control. If, for instance, the "Section1" subpart of the document in figure 1 is edited resulting in a new version, a new version of the document "ECSCW'93" is automatically created. The new version will contain the new version of "Section1" but will share unchanged parts with the old version of "ECSCW'93". Furthermore, the version control keeps track of and stores the deltas between the two versions of "Section1", i.e. the editing operations performed by the user for creating the new version from the old one.

This hierarchical fine-grained version control functionality is implemented as a basic functionality of the database server for hierarchical documents. A more elaborate description of the approach can be found in (Magnusson, Asklund, and Minör, 1993). Since it is fundamental in our editing model, it is also integrated into the other parts of the system, the hierarchical browser and the text editor. One reason is to support collaboration awareness, which we will expand on in the next section.

2.3 Collaboration awareness

In addition to handling versions and alternatives, our model supports collaborative editing by means of collaboration awareness. It is based on the hierarchical organization and the hierarchical version control functionality. Collaboration awareness is available in two ways in the system: by the evolution graph which is shared among all users and by the presentation of *active diffs*.

When a user enters the system to edit a document, the evolution graph gives a hint of the status of the document. The user can see who is editing it at the moment and what has happened since last time. In figure 1, for instance, one can see that there are two alternatives of the document and that the "Sten's work" alternative is edited at the moment (the editing key of this alternative is crossed over). If no other user is present when entering, the user can select his/her latest version for editing and use the system as any asynchronous editor. The system will automatically create a new version of the changed parts (and of the document) when the user starts editing them. It does not impose any overhead for dealing with versions compared to a conventional editor.

If other users are present one can still choose to asynchronously edit a version which is not in use by any other user. However, it is also possible to open a version currently edited by another user. In this case the system will create an alternative of the parts of the document the user edits. This alternative is edited independently of other alternatives of the same part. The alternatives may at a later time be merged. Since parts unchanged by any of the users still are shared between the alternatives, merging only has to take place for changed subparts of the document.

In order to make users aware of what other users actually are doing in the document, e.g. in two alternatives of a document part simultaneously edited by two users, the system provides active diffs. An active diff is showing the difference between two versions (or alternatives) of the document. The diffs are based on the actual edit operations performed, which are stored by the version/database handler as differences between versions. In this way fine-grained and accurate differences that reflect what modifications actually have been performed by the user can be presented. The diffs are presented in a semi-graphical form, at the presentation level quite similar to the presentation of diffs in Prep (Neuwirth, Chandhok, Kaufer, et al., 1992).

The "Section1" window in figure 1 shows a small example. The differences between the current and the previous versions are shown in the window used for editing. All insertions appear as underlined text and all deletions as "-" markers. The "-" markers may be expanded interactively in order to see the actual deletions. In the same way the underlined insertions may be collapsed to "+" markers. By collapsing all markers the user can see the text common to the two versions, by collapsing the minus markers and expanding the plus markers the text is viewed as the new version (as in figure 1), and finally by expanding the minus markers and collapsing the plus markers the text is viewed as the old version. Differences between parts of the document are shown in a similar way with the icons in the window titles marked as additions, deletions, and changes within a part (or one of its subparts).

By default the diff between the current version and its predecessor is shown. However, diffs between arbitrary versions and alternatives can be shown. This is done by selecting the versions (alternatives) in the evolution graph. In this way it is possible to see what has happened in the document since a certain version or in what way different alternatives differ. A view used in the merge situation shows the difference from the root version following the different alternatives (say A, B, etc.) with different markers. Diffs can also be shown for different alternatives currently being edited. If a user A sets up a diff between the alternative currently edited and an alternative edited by B simultaneously, the differences between the two versions will be shown continuously, i.e. all modifications performed by B will appear as markers in A's window. Notice that these diffs are shown on A's demand and A is free to turn off the diffs whenever wanted. The diff markers cannot be edited by A, A can only edit his/her own alternative. However, the markers make A aware of what B is doing and may contribute to avoiding conflicting changes which result in unnecessary resolving of conflicts during alternative merges at a later stage. The example with two users can be generalized to several users, e.g. using color coding of the markers in order to visualize who has changed what. Active alternative diffs may in this way be used for giving a "synchronous editing" view of the document, presenting a "what-if-I-merged" situation, useful for discussions, initial design, or brainstorming.

3. Properties of the model

The model for semi-synchronous editing we propose is based on the observation that a high degree of collaboration awareness can be obtained without actually sharing the document in the way synchronous collaborative editors usually do. Our model is basically based on an asynchronous style of editing using a traditional split-edit-merge approach for simultaneous editing. However, the fine-grained version control combined with the technique for active diffs adds collaboration awareness which comes close to pure synchronous editing. The fact that the user may choose the editing style from pure asynchronous editing to almost synchronous editing makes the approach very flexible. By selecting diffs the user may choose a style of editing which is in-between asynchronous and synchronous editing. The style can be changed while editing which allows several styles to be used within an editing session and a smooth transition between them.

The fact that simultaneous modifications of the same part of a document always are done in different alternatives and that changes made by other users merely are reflected by the active diffs have several technical consequences. One is that operations such as undo/redo of editing operations always are well defined. They may be applied to the local alternative in the same manner as in any asynchronous text editor. Another (more important) consequence is that no floor control is necessary. Neither locking, explicit or implicit, nor any turn-taking protocol to avoid conflicting operations is needed. If conflicting modifications are introduced they have to be resolved at merge-time. However, we expect that the use of active diffs will reduce the number of conflicts even though we do not have any empirical evidence for that at present. This expectation is supported by experience from the use of GROVE (Ellis, Gibbs, and Rein, 1991), which shows that conflicts are infrequent as a result of awareness of others' activities. Furthermore, we believe that this kind of "maximally lazy" consistency approach has the advantage that users need not be distracted by consistency issues, such as locking objects before editing or handling the floor between users, while performing the task.

Except for producing documents, the editing model can be used for commenting or reviewing. A reviewer can create an alternative of the document and add the comments. The author can at a later time (or actually at the same time if desired) set up an active diff to the reviewer's alternative and obtain the comments as markers in his/her own alternative. The author may then choose between just reading the markers as the document is revised in his/her own alternative and merging the two alternatives to a version containing the comments for further revisions.

The model is fairly robust against network failures. In case of a network failure during editing a user can continue editing. The collaboration awareness is lost and the editing has to continue asynchronously until the failure is repaired. We are also optimistic about that space and time requirements will scale up. The version/database manager is based on a backwards delta technique for storing versions which allows versions to be stored without maintaining multiple copies of the document. The time requirements are modest since the editing model does not rely on immediate distribution of editing operations among different users or the fact that they arrive at the different sites in a certain order. The active diffs are, of course, dependent on changes arriving within some reasonable time. The order of magnitude required is, however, seconds rather than fractions of a second.

A restriction of our model is that a document is hierarchically organized. This is due to the functionality of the database/version server which explicitly supports fine-grained version handling of hierarchical structures. In (Haake and Haake, 1993) version control for hypertexts in a cooperative system is described. Hypertexts put special demands on the version control server since it handles a general graph structure rather than a tree, which is not supported in our model. The main purpose of our model has been editing of hierarchical structures such as programs and documents. Based on our experience from the Mjølner project we do, however, believe that it may be applicable for other hierarchical structures, such as file-systems, mail-tools, spreadsheets, language grammars, and application-oriented languages.

4. Implementation status and future work

The editing model is the basic editing model in a tool for collaborative software development, which currently is developed within the object-oriented software development group at our department. The implementation is at the time of writing a fairly rudimentary prototype. It supports only editing of a program (or document) according to a grammar using the model presented in this paper. Some crucial parts still remain to be implemented in order to get a usable system. One is merging of alternatives based on the editing deltas maintained by the version server. Another is showing active diffs between several (more than two) alternatives which is necessary when several users are editing alternatives simultaneously.

Other areas we find interesting for future work are different kinds of merging. A merge can be made using an optimistic strategy, i.e. all changes made by any user, which is not in conflict with a change made by some other user, will be present in the version that is the result of the merge. A merge of documents not containing conflicting changes can be merged automatically using this strategy. However, modifications that are not in conflict in a technical sense may be in conflict semantically. A sentence added by one user may, for instance, contradict a sentence added by another user. A pessimistic strategy for merging should prompt the user for changes to the same document part (chapter, paragraph, sentence). It should also allow the user to control the result of the merge in case of a potential semantic conflict. Another reason for pessimistic merging is that the user may want to select different parts from different alternatives to the resulting merged version.

An area we have not yet dealt with is access control. This is particularly interesting for the active diffs. Since one user sees an active diff based on another user's editing operation deltas, it should be possible for the other user to work in a "private mode" where the editing deltas are not distributed to other users. Maybe it also should be possible to create versions and alternatives that are invisible to other users in the evolution graph.

Furthermore, the architecture of the distribution aspects of the system is a subject for further studies. At present the architecture is based on the client-server approach. The editors are replicated at each site and communicate with a central database/version server over a local area network. For collaborative editing over larger distances with unreliable communication this will not be sufficient.

Finally, we are interested in developing domain-specific support for software development where the editing model presented in this paper is included. This includes areas such as collaborative language-based editing, incremental compilation issues for collaborative systems, and collaborative debugging.

5. Conclusions

We have presented a model for semi-synchronous collaborative editing. The model relies on three fundamental concepts: hierarchical organization of documents, fine-grained version control, and presentation of active diffs. The hierarchical organization allows a document to be partitioned into separate parts, such as chapters, sections, and paragraphs, while the connection between the parts is maintained. The fine-grained version control mechanism maintains versions and alternatives of the document parts and the document itself. It supports collaboration using the split-edit-merge approach rather than sharing of the same material. Finally, the active diffs provide group awareness by showing differences between versions and alternatives as markers. In particular, the active diffs allow presentation of diffs between alternatives while different users edit the alternatives, resulting in a continuous updating of the diff presentation.

By integrating the above functionality, an editor that fills the gap between asynchronous and synchronous editing is obtained. The editor may be used asynchronously, synchronously, or in a mode in-between. The mode can easily be changed during an editing session and may be different for different users working simultaneously.

The aim has been to obtain an editing model that support the way groups actually work in collaborative editing, using different writing strategies during different activities and phases of a project. In order to obtain a usable system for some application area, it has to be extended with domain-specific support, e.g. for authoring or software development.

Acknowledgements

The authors wants to thank all the members of the software development research group at Department of Computer Science, Lund University, for stimulating discussions which have contributed substantially to the work presented in this paper. In particular, we want to thank Ulf Asklund and Torsten Olsson, who are involved in designing and implementing the fine-granular version/database server, the text editor, and the active diffs.

The work presented in this paper was supported in part by NUTEK, the Swedish National Board for Industrial and Technical Development.

References

Dourish, P. and Belotti, V. (1992): “Awareness and Collaboration in Shared Workspaces”, *Proceedings of CSCW’92, ACM 1992 Conference on Computer-Supported Cooperative Work*, Toronto, 1992

Ellis, C.A., Gibbs, S.J., and Rein, G.L. (1991): “Groupware: Some Issues and Experiences”, *Communications of the ACM*, 34(1), January, 1991

Greenberg, S. and Bohnet, R. (1991): “GroupSketch: A Multi-User Sketchpad for Geographically Distributed Small Groups”, *Proceedings of Graphical Interface ‘91*, Calgary, 1991

Gustavsson, A. (1990): *Software Configuration Management in an Integrated Environment*, Licentiate thesis, Dept. of Computer Science, Lund University, Lund, 1990

Haake, A. and Haake, J. (1993): “Take CoVer: Exploiting Version Support in Cooperative Systems”, *Proceedings of INTERCHI’93, 1993 Conference on Human Factors in Computing Systems*, Amsterdam, 1993

Hedin, G. and Magnusson, B. (1988): “The Mjølner Environment: Direct Interaction with Abstractions”, *Proceedings of ECOOP’88, 2nd European Conference on Object-Oriented Programming, Lecture Notes in Computer Science*, vol. 322, Springer-Verlag, 1988

Haake, J.M. and Wilson, B. (1992): “Supporting Collaborative Writing of Hyperdocuments in SEPIA”, *Proceedings of CSCW’92, ACM 1992 Conference on Computer-Supported Cooperative Work*, Toronto, 1992

Knudsen, J.L., Löfgren, M., Madsen, O.L. and Magnusson, B. (1993): *Object-Oriented Development - The Mjølner Approach*, Prentice-Hall, 1993, to appear

Knister, M.J. and Prakash, A. (1990): “DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors”, *Proceedings of CSCW’90, ACM 1990 Conference on Computer Supported Cooperative Work*, Los-Angeles, 1990

Magnusson, B., Asklund, U., and Minör, S. (1993): “Fine-Grained Revision Control for Collaborative Software Development”, Tech. Report LU-CS-TR:93-112, Dept. of Computer Science, Lund University, Lund, Sweden, 1993

Magnusson, B., Minör, S., Hedin, G., Bengtsson, M., Dahlin, L., Fries, G., Gustavsson, A., Oscarsson, D., Taube, M. (1990): “An Overview of the Mjølner Orm Environment”, *Proceedings of the 2nd International Conference TOOLS (Technology of Object-Oriented*

Languages and Systems), Paris, 1990

McGuffin, L.J. and Olson, G.M. (1992): “ShrEdit: A Shared Electronic Workspace”, Cognitive Science and Machine Intelligence Laboratory, Tech. report #45, University of Michigan, Ann Arbor, 1992

Neuwirth, C.M., Chandhok, R., Kaufer, D.S., Erion, P., Morris, J. and Miller, D. (1992): “Flexible Diff-ing In a Collaborative Writing System”, *Proceedings of CSCW’92, ACM 1992 Conference on Computer-Supported Cooperative Work*, Toronto, 1992

Neuwirth, C.M, Kaufer, D.S., Chandhok, R. and Morris, J. (1990): “Issues in the Design of Computer Support for Co-Authoring and Commenting”, *Proceedings of CSCW’90, ACM 1990 Conference on Computer Supported Cooperative Work*, Los-Angeles, 1990

Posner, I.R. and Baecker, R.M. (1992): “How People Write Together”, *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Vol. 4, Hawaii, 1992.

Tichy, W.F. (1988): “Tools for Software Configuration Management”, *Proceedings of the International Workshop on Software version and Configuration Control*, Grassau, Germany, 1988