Group Awareness Using Fine-Grained Revision Control

Torsten Olsson

LU-CS-TR:94-132

Also published in Proceedings of NWPER'94, Nordic Workshop on
Programming Environment Research, Lund, Sweden, June 1994.

Department of Computer Science
Lund University

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

Group Awareness Using Fine-Grained Revision Control

Torsten Olsson

Department of Computer Science, Lund University
Box 118, S-221 00 Lund, Sweden
e-mail: Torsten.Olsson@dna.lth.se

Abstract. This paper describes work on providing support for *teams* of software engineers in software development environments. Particularly integrated support for merging and providing group awareness among team members is discussed.The paper gives an brief overview of the project, an introduction to the revision control approach, and presents the notion of active diffs, which is central to the merging and group awareness support.

1 Introduction

This paper describes our approach and tool for supporting cooperative development of text document and program source code. All features described in this paper are not yet fully implemented but we have done enough to be sure that it is possible to fully implement what is described here.

In later years software systems have become more and more complex. To develop and maintain such large systems, an increasing number of people working together are required. This calls for tools that support development where many people work on the same set of documents [1] at the same time. This tool should not only give the users awareness of what others are doing but also guarantee that the documents are kept consistent, i.e. keep users from interfering with each others work and keep track of the versions and revisions of the documents. Research in this area can be divided into three main areas:

- Communication. Tools that let the user talk to, leave messages to, set up conferences with, and share the screen with other users.
- Project and process management. Tools that help planning, coordinating, and keeping track of the evolvement of the project.
- Document sharing and organization. The handling of the large amount of documents, keeping related documents consistent (e.g. Unix “make” keep the object code and executable programs up to date with the source code), or keeping track of versions and revisions and help the user from interfering with each others work (e.g. Unix “RCS” utilities [Tic85]).

1. In this paper we use the term *document* for everything that are produced in the development process e.g. source code, design documents, and documentation.

There has been many attempts to address one or more at a time, e.g. [GB91], [HW92], [MO92], [NTBB92], and [NKCM90].

Our work has been focused on document sharing and organization. The system is built on a fine-grained revision control system of a hierarchical organized document. This means that all changes to the document's structure or textual contents, are stored and the development history of the document can be examined in detail. Every time a document is checked in, a new revision of that document is created. The connections between revisions are shown in a *revision graph*. When there is a straight development line where each new instance succeed the previous we call them *revisions*. If, on the other hand, a document is split up in two or more development lines they are called *alternatives*.

An alternative is created every time two[1] developers start working on the same part of the document at the same time, i.e. there are no locking mechanisms to ensure consistency[2]. Instead every developer is free to work on all parts of the document without restrictions. This approach is called optimistic checkout. Thanks to the fine-grained revision control system and the hierarchical structure of the document it is unlikely that collisions between the work of developers will be frequent. But if it happens, alternative document parts are created and can later on be merged together. A revision graph shared among all users reflects changes to the document and is used to navigate trough the history of the document. It is also possible to view differences between two alternatives. This is especially interesting when there are people working on both alternatives and they want to se what the others are doing. Another case when it is interesting is prior to doing a merge.

This paper is focused on the fine-grained revision control system that keep track of the updates to the text document and a mechanism for viewing the dynamic differences between two alternatives under development, the *active diffs*. Actually there are two slightly different kinds of diffs, one that handles the structure or hierachy of the document and one for the pure text as outlined in this paper. The basic ideas are the same but there are differences in both implementation and user interface. A description of the structure diffs can be found in [Ask94].

The paper is organized as follows: in the next section we give an overview of the system, then there is a section that explore the features and some of the implementation of fine-grained revision control and active diffs.

2 System Overview

The central ideas behind our system are:

- Hierarchical document structure. The basic structure of a document in the system is hierarchical. This fits nicely on both text and programs. Lets take a normal text like this one as an example. The "root" in the structure

1. When discussing alternatives and merge we often write "two developers" instead of "two or more developers" in order to make the text more readable.

2. Consistency means that the changes made in two alternative don't interfere with each other when merged together.

is the whole document. It is divided into chapters which in turn is divided into sections and so on.

- Fine grained revision control. This means that each level in the document is revision controlled and all edit operation are kept track of. This is important because the granularity of the revision control tells in what detail the evolution of the documents can be studied in retrospect. It is also the foundation for our approach to group awareness and merging.
- Optimistic check-out. We believe that most work of a developer do *not* interfere with the work of any others. Therefore there are no restriction on which part of a document a developer can start working on. Sometimes this leads to conflict when two or more developers changes the same document at the same time. This situation is detected on check-out and an alternative revisions are created.
- Advanced support for merge. It is sometimes necessary to merge two alternatives into one singe revision of the document e.g. when alternatives have been created by the optimistic check-out system. Our system supports detection of conflicts and automatic merge if no such conflicts are detected.
- Group awareness. The main source of group awareness is the revision graph. It allows people see when others are checking in and out changes. In addition a feature, active diffs, makes it possible to track the development of a document. It is possible to examine what is done in an other alternative by an other user compared to your alternative.

These features together with the editing model described below provides the fundament for a flexible and extensible tool that supports different working styles.

There are two basic ways of interacting when working on a common document. In the fist case two or more people are working together on the same part of the document. They want to share the same view of the document, perhaps even share the contents of the screen. The document is used as a basis for discussion when interacting on a "metalevel" e.g. using the telephone. Some tools even have support for this metalevel discussion built in e.g. GroupSketch [GB91]. This type of interaction is called synchronous.

In the second case we have individual assignments that are mostly independent from the work of others. In this case you have your independent part of the document and you don't want to se what are done by the other parts. When done, the updated part of the document is made available to others to use. This kind of interaction is called asynchronous.

It is clear that a system supporting many people working together on the development of a document has to be able to support both synchronous and asynchronous editing in some form. In our system we have tried to meet both needs but concentrated on the asynchronous editing because it appears to be the most common one. We have then added some features in the direction of synchronous editing mainly to emphasize the feel of group work. For example is the revision graph synchronously updated and the work of other developer can to some extent be viewed using active diffs.

The revision graph in combination with the fine-grained revision control makes it possible to view any revision of a document. It is also possible to show the differences between any revisions or alternatives. The only restriction is that the development lines have one common ancestor somewhere in the revision tree.

In figure 1 a screen dump of the system is presented and followed of some explanations.

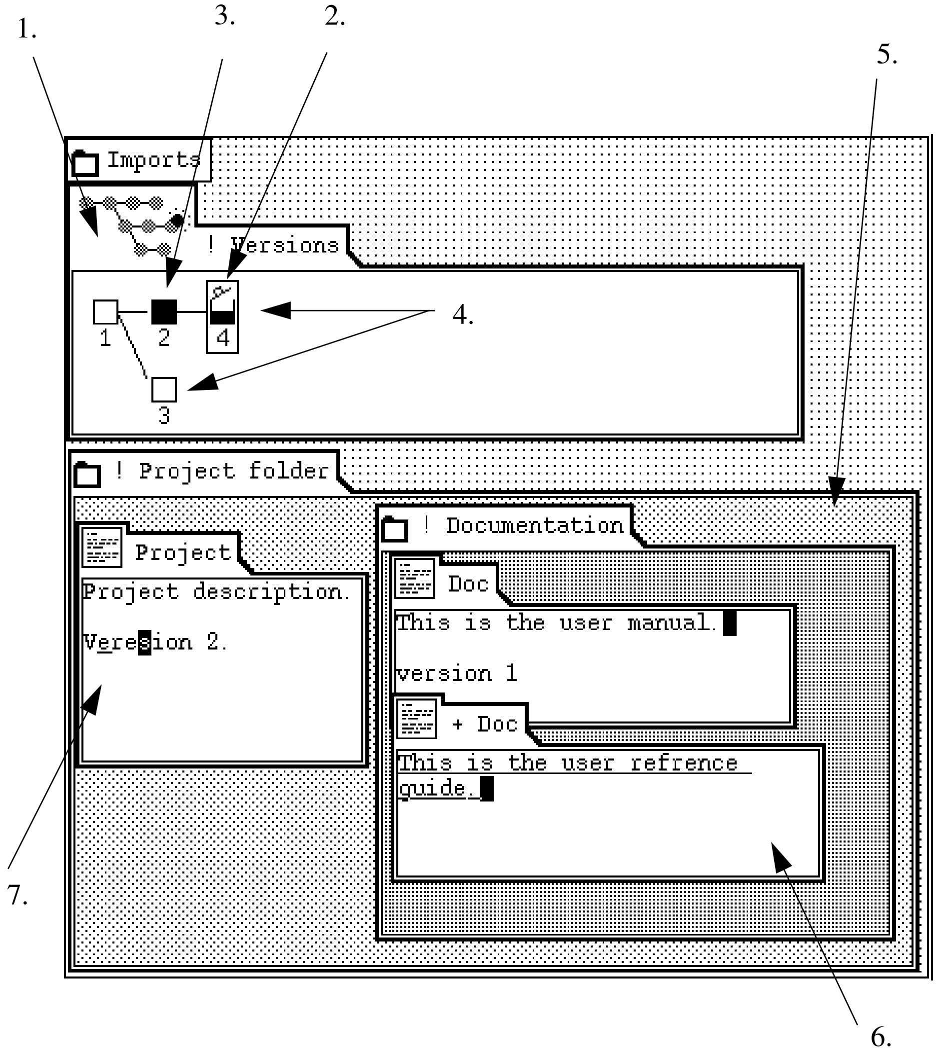

Figure 1 *Our system in use.*

1. The revision graph. Each square represent a checked in revision and the lines between the squares, read from left to right, shows the relationship among the revisions, e.g. revision 3 originates from revision 1 of the document.
2. Symbolize the current revision, i.e. the one being edited.

3. Symbolize the reference revision, i.e. the revision to which all shown diffs are related.
4. Alternative development lines from a common base document. The creation of an alternative can be either deliberate or accidental. In the fist case the alternative is created on purpose, e.g. to examine for example different way to implement a feature. In the second case the alternative is created by the system when two users start working on the same part of the document at the same time.
5. The hierarchical structure of the document is reflected in the window structure. The "Project folder" is the root of the hierarchy . In this folder there are one document, "Project", and one folder , "Documentation". These represent the two nodes below the root node. The node "Documentation" in turn contains two documents
6. This document part is added in the current revision or to be more specific, somewhere after the reference revision. This is marked with a "+" in the window header . The text in the window is underlined which means the same for text as the "+" marking does for windows, i.e. document parts.
7. In this window the user is editing the contents of the window as text, currently correcting the spelling of the word "version". Note that the inserted "e" is underlined and when the misplaced "e" is removed it will leave a "marker" indicating that something has been removed. This marker can be expanded to determine what was removed. Also note that the window has no "+" marking because it existed in the reference revision, it is only the contents of the window that is changed. A similar approach is given in [NCK+92].

The system is partly based on previous work in the Mjölner project. That project was mainly concerned with object-oriented software development [KLMM93]. One of the results, the Mjölner Orm Environment [MMH90] had a limited support for cooperative development through it is configuration manager [Gus90]. In the next chapter we will explore the possibilities of the fine-grained revision control and the active diffs that are using these as a fundament.

3 Active Diffs Based on Fine-Grained Revision Control

One of the most important parts of the system is the fine-grained revision control system and the active diffs that are using them. In this chapter the terms will be explained and the possibilities will be explored.

In the system any update is saved as a difference between the current and the previous revision. These deltas [1] are based on the actual edit operations performed rather than on a comparison of the two revision as in the Unix diff program. In this way , fine-grained diffs truly reflecting the changes in the document can be presented to the user . In order to do this the editing tool have to be integrated in the environment and be able to generate and show diffs.

1. In this text the term *diff* is used for differences between documents that are shown to the user in contrast to deltas that are the internal representation e.g. the representation on disc.

Another important use of the fine-grained deltas is when merging alternatives of a document. They are then used to determine if there are conflicts, i.e. if the same part of the document have been edited in both alternatives. In that case the user must decide how to resolve the conflict. Hence, it is not possible to do an automatic merge on that part of the document. Note that not only the text but the whole structure is handled by the fine-grained revision control mechanism. This means that the system at all times can determine the smallest possible parts for a merge and leave the rest unchanged.

One important feature, similar to merge, is active diffs. An active diff is the result of a "automatic virtual merge" between an active alternative and an alternative being developed by another user . The result is a document that shows the result of a merge without actually having to perform the merge. This is a mechanism that allow the user to se if and how his work interfere with the work of another user and thus provide group awareness.

In the rest if this chapter we will examine the properties of the fine-grained revision control and the active diffs.

3.1 Fine-Grained Revision Control

The idea behind the fine-grained revision control is that it shall be possible to exactly determine what the difference are between two revision of the same document. T o accomplish this, the basic edit operations on the text and structure have to be stored. It is not possible to determine afterward as the difference between two document the way the unix "diff" command works. One side effect is that it is not necessary to save the whole document for every revision. It is enough to save the document once and for every revision save the difference to the next, see figure 2. In this way we have more information about the development of the current revisions of the document and disc space is saved.

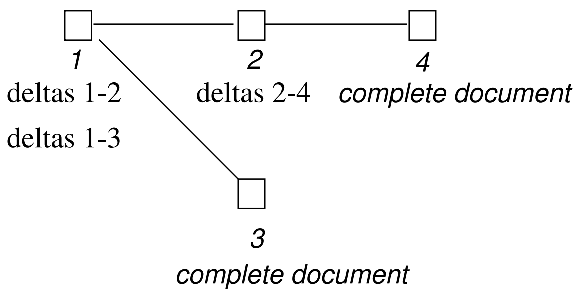

Figure 2 *Storing of document using fine-grained revision control*

To be able to store the edit operations as deltas, an integrated environment where editors for the structure and text work hand in hand with the storage mechanisms is necessary . The deltas must have certain properties to be able to perform the tasks assigned to them. They must be able to:

1. Store the basic edit operations, insert and delete.

2. Be combined with other deltas, i.e. when viewing the difference between revision 1 and 4 in figure 2 two separate sets of deltas must be combined together in order to form the resulting set of deltas.
3. Be applied backwards, i.e. it must be able to be applied in both directions in order to show diffs along a development line and create diffs between alternatives. Consider figure 2. In order to show revision 1 of the document the "delta 1-3" must be applied to the complete document in revision 3 i.e. the deltas are applied to a revision in order to get the previous one. If, on the other hand, the purpose is to show the difference between revision 3 and 4 it can be done starting from revision 4, applying "delta 2-4", "delta 1-2", and "delta 1-3" in that order . Note that "delta 1-3" now is used in the opposite direction.

It turned out that a rather simple data structure was able to meet those needs. The structure of the deltas is based on insertion and deletion of text fragments. Each operation stores type, the inserted or deleted text, and position in the document based on lines and rows. The reason for the text to be included in both types of operations is point 3 above.

The total size of the all the deltas in the revision graph is hard to estimate because it depend on how often users changes their minds, i.e. if they do a lot of text insertion in one revision and then delete the text in the next. One minimum case is when there are only insertions. The user know what to write and never deletes anything. In that case the total amount of disk space is about 2*(size of resulting document). This is the minimum. Then for every deletion a delta that is about the size of the deleted/inserted text is stored. Thus the total amount of disc space needed is more correlated to the number and type of revisions than to the size of the resulting document. To be able to handle large revision graphs with a large number of fine revision steps there is need for some mechanism that collapses several nodes into one. This not only make the revision graph easier to overview but also reduces discspace.

The time used to open the newest version of the document is momentanous. This is due to the fact that the document is divided into small parts that are loaded "lazy" and that the complete document is stored at the leaf in the revision graph. We have not yet been able to measure speed on loading older versions but we don't expect it to be any problem except in extreme cases.

The internal representation in the editor is under development. It is designed but not yet fully tested.

3.2 Active diffs

The active diffs are nothing but an automatic merge that is shown but not saved. This may sound a little confusing but the idea is quite simple. If two developers are changing the same document part, alternative development lines are created. To get back to one development line, a merge have to take place, see figure 3.

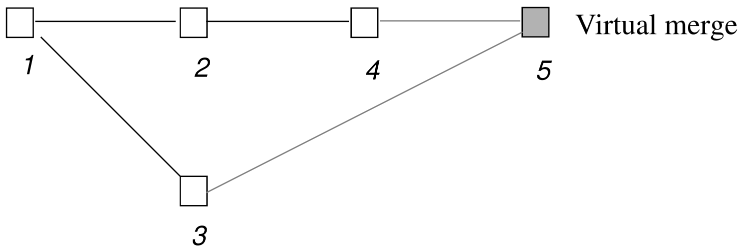

Figure 3 *The principle of the active diffs.*

In the merge process the differences in the two alternatives are compared and the two documents are combined on basis of this. This can be done automatically if there are no conflicting changes. The possible conflicts between text document that are not going to be handled automatically are:

- When one alternative have corrected a sentence and the other have removed it.
- When text has been inserted on the same spot in both documents. In this case it is impossible to determine which one to use or if both are relevant, in which order they are meant to appear.
- When areas of deleted text overlap on the two alternatives.

This conditions need to be examined by a human who can decide what to do. In order to perform straightforward merge it is then essential to minimize or totally eliminate the number of conflicts. This is the most important use of active diff.

Now reconsider figure 3 and imagine one person working on revision 4 and one on revision 3. These are alternative development lines originating in revision 1. The intension is to merge the alternative back together at a later stage and the developers want to be sure that their work don't interfere to much in order to minimize conflicts during the future merge. In order to do this they set up a virtual merge between the two alternatives. The system then shows what would be the result if an automatic merge would be performed continuously. This means that the developers edit in a document that at every moment reflects the changes that are made of all of them e.g. revision 5 in figure 3. One problem that is not yet solved is how to present the conflicts to the users.

4 Summary and Future Work

In this paper an approach to a tool supporting cooperative work has been presented and some if the ideas behind it has been studied in more detail. The main ideas are:

- Hierarchical document structure.

- Fine grained revision control.
- Optimistic check-out.
- Advanced support for merge.
- Group awareness.

The aim has been to create a tool that are flexible enough to allow different writing strategies during different phases in the development cycle. The first version of the system is text based. This makes it useful for the development of text document as well as source code for software systems. We have not yet finished the implementation and thus not tested the whole system on a larger project but on basis of what we have done it seems promising. There are several interesting areas to explore further. Graphical representations in program design are becoming more and more frequent. Therefore it is interesting to explore the possibilities of revision control and merge of graphical representations. Another area is to extend the merge capabilities to source code where also semantic relations are considered when deciding if changes are interfering with each other. For instance, if a procedure declaration is changed in all references to this procedure are considered during a merge.

Acknowledgment
I want to thank all the members of the software development research group at Dept. of Computer Science, Lund University, for stimulating discussions which have contributed substantially to the work presented in this paper. In particular, I want to thank Sten Minör for constructive comments on earlier drafts of this paper.

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

References

[Ask94] Asklund, U: "Identifying Conflicts During Structural Merge", *Proceedings of the Nordic Workshop on Programming Environment Research*, Lund, 1994.

[GB91] Greenberg, S. and Bohnet, R.: "GroupSketch: A Multi-User Sketchpad for Geographically Distributed Small Groups", *Proceedings of Graphical Interface '91*, Calgary, 1991

[Gus90] Gustavsson, A., "Software Configuration Management in an Integrated Environment", *Licentiate thesis, Dept. of Computer Science, Lund* University, Lund, 1990.

[HW92] Haake, J.M. and Wilson, B. : "Supporting Collaborative Writing of Hyperdocuments in SEPIA", *Proceedings of CSCW'92, ACM 1992 Conference on Computer-Supported Cooperative Work*, Toronto, 1992.

[KLMM93] Knudsen, J.L., Löfgren, M., Madsen, O.L. and Magnusson, B.: *Object-Oriented Environments - The Mjølner Approach*, Prentice-Hall, 1993.

[NTBB92] Kaplan, S.M., Tolone, W.J., Bogia, D.P., and Bignoli, C.: "Flexible, Active Support for Collaborative Work with ConversationBuilder", *Proceedings of CSCW'92, ACM 1992 Conference on Computer-Supported Cooperative Work*, Toronto, 1992.

[MMH90] Magnusson, B., Minör, S., Hedin, G., et al., An Overview of the Mjølner

Orm Environment, *Proceedings of the 2nd International Conference TOOLS (Technology of Object-Oriented Languages and Systems)*, Paris, 1990.

[MO92] McGuffin, L.J. and Olson, G.M.: "ShrEdit: A Shared Electronic Workspace", *Cognitive Science and Machine Intelligence Laboratory, Tech. report #45*, University of Michigan, Ann Arbor, 1992.

[NCK+92] Neuwirth, C.M., Chandhok, R., Kaufer, D.S., Erion, P., Morris, J. and Miller, D.: "Flexible Diff-ing In a Collaborative Writing System", *Proceedings of CSCW'92, ACM 1992 Conference on Computer-Supported Cooperative Work*, Toronto, 1992.

[NKCM90] Neuwirth, C.M, Kaufer, D.S., Chandhok, R. and Morris, J.: "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.

[Tic85] Tichy, W.F., RCS - a system for revision control. *Software-Practice and Experience*, 15(7):637-634, July 1985.