Collaborative Editing - distribution and replication
of shared versioned objects

Boris Magnusson
Ulf Asklund

LU-CS-TR:96-162

Also presented at European Conference on object Oriented Programming
1995, in Workshop on Mobility and Replication, Aarhus, August 1995

Department of Computer Science
Lund Institute of Technology
Lund University

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

Collaborative Editing – distribution and replication of shared versioned objects.

Boris Magnusson and Ulf Asklund
Dept of Computer Science, Lund University
Box 1 18, S-221 00 Lund, Sweden
e-mail: {Boris|Ulf} @dna.lth.se

Position paper . We take the starting point in a system for collaborative editing in a distributed environment. It includes integrated support for fine-grained version control and configuration management. We describe how such object can be migrated to independent sites, edited, and then merged into one object with several alternatives. We also show that our model is meaningful to understand other applications.

1 Background

The original motivation for our work was to support teams of software developers working together , possibly in a distributed environment. In this situation the need for version control turned out to be even more crucial than usual. Versions and alternatives are fundamental when several people are working simultaneously on the same data. Questions like *What change did I make? What are the difference between our versions?* and *What changes have been made to the systems during my vacation?* arises. In order to make it easy to answer such questions we came very quickly to the conclusion that a collaborative editing system for software needs a tightly integrated version control system with an intuitive user interface. In the system we have built (COOP/Orm) we have made a number of nontraditional design decisions:

• Version control is part of the fundamental storage model which means that all processors, including editors, must understand (and maintain) versions and deltas.

• Versions can naturally not change, which means that an object can only be changed through addition of versions and alternatives. An object is considered a collection of immutable versions of its contents.

• Lock on check-out does not scale up to a large number of users or to large software systems. We therefor allow changes at any time (creating alternatives) and provide support for merging alternatives.

• Hierarchy is a common structure in documents (in programs, papers, books etc.) and is supported in our storage model.

The system is designed for a multi Client/multi Server situation. The Client-Server setup takes care of serializing access from the clients to the file in the filesystem where the information is stored in the end. The Server/Server communication is the interesting aspect from distribution point of view . This protocol supports replication of the objects at different sites. It is motivated by the possibility of network failures and by performance. From the users point of view the Client/Server protocol provides the important functionality while the distribution provided by the Server/Server protocol should not be visible.

In the rest of this position paper we will try to explain how the approach of representing the full sequence of immutable versions of an object actually supports migration but still provides a useful distributed system for an interesting class of problems.

2 Supporting replication through integrated version control

Replication at different distributed sites is in our situation motivated primarily by performance and availability in case of failures. The problem is of course that if the replicated data is changed at several sites it is in general very hard to get back to one consistent copy of the object again. This is hard for changes to plain data attributes, but soon unmanageable when changes to the structure of the object are allowed. Simpler cases can be considered. If, for example, an object is not changed there is no problem with replication, but in general this is not an acceptable restriction. If there are only additions to the replicated objects, the problem is smaller than when general changes are allowed. If the additions are unrelated with respect to order , or can be ordered automatically , bringing the replicas back is simple. Our approach can be viewed as taking this additions only approach and applying it to an object with all its versions rather than a particular version of the object.

The view that an object is representing its full history (not just a snapshot as is common) is fundamental here. This view means that the object can only be modified by creating a new version, that is an addition. Note that the new version of the object might be defined after deletion of some part of the object from an earlier version of the object. Also in this case the object is still growing with another version.

The word 'merge' can in our system be used for operations on two different levels, on the version history of the object, and on the contents of the object itself. Normally our servers are in contact and communicate changes to replicated objects, made at the different sites. There can, however , also be for example network failures which means that replicated objects can drift far away in their version history and needs to be synchronized when the failure is removed. In this case, each replica might been extended with one or more new versions. These additions are not related and thus the two version histories can automatically be 'merged'. The layout of a graphical representation of the version graph might have to be changed to accommodate the 'other ' versions as well, but this is only a presentation change, having no semantic meaning.

The second meaning of 'merge' here is to combine two alternative versions of an object to a single new version. This merge problem is not unique to replication, but will appear also in a single server situation, and indeed also with a single user working on parallel development tracks. Our support for this builds on storing the edit operations, utilizing the hierarchical representation to recognize large portions of a structured object that are unchanged and default rules for suggesting a merge. It also detects conflicts that has to be solved by the user doing the merge. The only addition to this scheme that comes from replication is that an alterative can be created without intent and a future merge might have to be done. We have described elsewhere the details of our merging facilities and compact storage representation [MM93, MAM93, Ask94, Ols94, MG95, MA95].

3 Discussion

Our model has been developed to support cooperative work. The model seems to work well as a basis for collaborative editors. The ability to support simultaneous development is here very valuable since there is a tendency to move away from traditional lock on check-out policies in software development situations. The experience seems to be that conflicting changes are fewer than feared. Other mechanisms, such as policies for sharing the work, seems to limit the need for several persons to actually change the same document at the same time. On the other hand, not providing the possibility causes sever problems and in the end the version control system (if insisting on locking) is cheated. Our strategy is to provide powerful support for merge to make this step easy when it is needed. Replication in this situation is thus feasible.

There are probably other distributed applications where the same scheme could be used. At first it would appear as unacceptable to use this approach for a bank – a merge of replicated versions of a bank account could result in a overdraft of the account. On the other hand this seems to be what banks actually do when allowing withdrawal of money from automatic teller machines when there is no connection to the bank (at least most Swedish banks allow this, but limit the amount of money that can be withdrawn and thus the risk).

Another application that comes to mind is airline reservation systems that seems to be allowing a certain amount of replication and distributed changes. A similar situation arrises when booking a flight, it appears to have free seats, accepts preliminary bookings, but a little later reservations are refused since the flight is suddenly full. This can be seen as a merge situation with a detected conflict. The default behavior here is to refuse to do the merge when a conflict is detected.

In conclusion there thus seems to be other applications that use a similar approach. Approaches that can be understood in our model.

Another interesting question is what constitutes a transaction in our model. We work with changes to the version history of an object, rather than changes to a particular version of the object. The answer is thus that we work with extremely long transactions, being the creation of a new version of the object which has a well defined start and finish. Other users can be engaged in other long transactions, creating other versions of the object at the same time. We even (plan to) allow users to look at versions as they are being create by other users (but not change them of course). It is here interesting to look at such versions, to compare them with finished or versions under creation and to find out if there are any conflicts although the version is not yet finished. In our model these long transaction, are thus not atomic at all. The necessary protection from havoc is provided by giving each user a version of his own to work with. The shared information is really the version history, but also here atomicity is not needed since only additions are made.

References

[Ask94] Ulf Asklund. Identifying Conflicts During Structural Merge. In Magnusson et al. MHM94.

[Ced93] Per Cederqvist. Version Management with CVS. Available from info@signum.se, 1993.

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

[HH93] Anja Haake and Jörg M. Haake. Take CoVer: Exploiting Version Support in Cooperative Systems. In *Proceedings of INTERCHI'93*, ACM Press, Amsterdam, The Netherlands, April 24-29 1993. Addison Wesley.

[HM88] G. Hedin and B. Magnusson. The Mjølner environment: Direct interaction with abstractions. In S. Gjessing and K. Nygaard, editors, *Proceedings of the 2nd European Conference on Object-Oriented Programming (ECOOP'88)*, volume 322 of *Lecture Notes in Computer Science*, pages 41–54, Oslo, August 1988. Springer-Verlag.

[Kat90] Randy H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4), December 1990.

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

[LvO92] Ernst Lippe and Norbert van Oosterom. Operation-based Merging. In H. Weber, editor, *SIGSOFT'92 Proceedings*, Tyson's Corner, Va., December 1992. ACM. SIGSOFT Software Engineering Notes, 17(5).

[MAM93] Boris Magnusson, Ulf Asklund, and Sten Minör. Fine-Grained Revision Control for Collaborative Software Development. In *Proceedings of ACM SIGSOFT'93 - Symposium on the Foundations of Software Engineering*, Los Angeles, California, 7-10 December 1993.

[MA95] Boris Magnusson and Ulf Asklund. XXX: A Model for Fine Grained Version Control of Configurations. Draft report, Lund University 1995.

[MG95] Boris Magnusson and Rachid Guerraoui. Support for Collaborative Object-Oriented Development. Draft report, Lund University 1995.

[MHM94] Boris Magnusson, Görel Hedin, and Sten Minör, editors. *Proceedings of the Nordic Workshop on Programming Environment Research*, Lund University of Technology. LU-CS-TR:94-127, Lund, January 1-3 1994.

[MM93] Sten Minör and Boris Magnusson. A Model for Semi-(a)Synchronous Collaborative Editing. In *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, Milano, Italy, 1993. Kluwer Academic Publishers.

[MMH+90] Boris Magnusson, Sten Minör, Görel Hedin, et al. An Overview of the Mjølner Orm Environment. In J. Bezivin et al., editors, *Proceedings of the 2nd International Conference TOOLS (Technology of Object-Oriented Languages and Systems)*, Paris, June 1990. Angkor.

[Ols94] Torsten Olsson. Group Awareness Using Fine-Grained Revision Control. In Magnusson et al. MHM94.

[Tic88] Walter F. Tichy. Tools for software configuration management. In *Proceedings from International Workshop on Software Version and Configuration Control*, Grassau, Germany, February 1988.

[TJ88] Dave Thomas and Kent Johnson. Orwell: A Configuration Management System For Team Programming. In N. Meyrowitz, editor, *Proceedings of OOPSLA'88*, San Diego, Ca., September 25-30 1988. ACM. SIGPLAN Notices, 23(11).

[Wat] Gray Watson. CVS Tutorial. Available from gray.watson@antaire.com.