

AN EXPERIMENT IN “OFF THE SHELF” COLLABORATIVE EDITING

Mikkel G. Brun and David Redmiles

Department of Information and Computer Science

University of California, Irvine

{mbrun, redmiles}@ics.uci.edu

Abstract

Various special-purpose tools attempt to support users in collaborative writing and editing tasks. Our approach has been to experiment with general-purpose tools and standards, familiar to larger communities, to determine what degree of support for collaborative editing “off the shelf” products could provide. The work focused on adapting relational database tools to coordinate users’ work and representing shared documents using SGML (Standard Generalized Markup Language). Thus, it is our hypothesis that “off the shelf” databases can be used as “collaboration servers” in a platform independent client / server environment of applications within different domains. A prototype simulating the intended behavior of the client / server architecture has been implemented to demonstrate the concept and reveal problems in the model. This paper summarizes our findings to date and discusses issues for future research.

Introduction

In both the research and commercial world, new applications are emerging to support collaborative work and, in particular, collaborative writing. However, instead of introducing entirely new tools, Grudin suggests that support for collaboration be incorporated in existing, single user applications [Grudin 94]. For example, in the case of word processing applications, collaborative writing features would slowly be added to existing word processors. Advantages are that existing tools are familiar, perceived as reliable, and are readily available to users. Ideally, different users collaborating on a single writing effort would each use the editing tool they most preferred.

This ideal assumes that cross platform collaboration will eventually be a reality. We argue that it is better to work toward this goal sooner rather than later. Furthermore, our approach to collaborative support is very pragmatic. In the same spirit as allowing users to work with familiar, editing tool, we want to devise a model that will allow developers to integrate support for collaboration through “off the shelf” applications. By “off the shelf”, we mean that the collaboration support is composed of defacto industrial standards and commercial products.

We have found client / server relational database systems to be the engines we are looking for. They are reliable and widely used in the developer community. By using an “off the shelf” engine we eliminate the need of having to deal with concurrency control, data integrity and other low level problems. A prototype implementation examined the minimal requirements for adapting a client / server relational database product to allow collaborative editing of documents represented using an SGML (Standard Generalized Markup Language) format. After a brief description of the domain of collaborative writing, we describe the implementation of the client / server architecture followed by a discussion of the lessons learned.

The Domain of Collaborative Editing

We investigate the possibility of adding “off the shelf” collaboration support to word processing. Academic co-authoring poses special problems usually not encountered in other environments. Colleagues often work in different institutions at great distance. They have to cope with incompatible editing systems which makes it difficult to exchange formatted files without loss of formatting. The incompatibilities often lead to a situation where one co-author takes care of formatting and incorporation of the other co-authors work [Ellis et al. 94, McGuffin, Olson 92].

Developers have tried to overcome some of these problems by implementing collaborative editing systems in which users work on shared documents. The available collaborative writing systems are very diverse. Most of them have been developed in universities by researchers in order to research the impact on work group collaboration, co-authoring and group performance. The groups have focused on quick development of applications that could be tested, rather than spending time on developing viable representations for shared text. The systems were therefore not compatible and could thus not satisfy individual preferences, and larger documents were eventually at some time moved to a more advanced editing system. Now that the field has gained some initial experiences with the concept it is time for commercial developers to give their contributions.

A Client / Server Architecture for Collaboration

The “SGML/RDB” model is our example of “off the shelf” collaboration. The model specifies the interface between word processing applications and an SQL-based relational database. The model furthermore specifies a structure on the relational database and describes the atomic operations on that structure issued by the word processing application.

Databases, Text & Standard Generalized Markup Language (SGML)

Imagine that several users collaboratively edit a shared text using word processing applications from different vendors. The text would have to be stored in a standardized format on a “collaboration server”. The SGML standard is an excellent standard intended for the representation of single-user documents. A fortunate attribute of texts following the standard is that they are very well suited for storage in relational databases. By combining two established representations of data, a multi-user relational database and text marked up using SGML, do we attain a powerful back-end to collaborative editing-systems. In the following we will refer to this model for text representation as the SGML/RDB model.

We envision a model where word processing applications serve as front-ends to database-servers (See Figure 1). Any text editor which follows the standard will be able to edit a given text. Users can thus use their favorite word processor for both conventional single-user documents as well as multi-user documents. A front-end (user) can simultaneously access documents on several servers. A user with simple editor will of course only be able to view the text in a basic format whereas a user with an advanced layout program will be able to view the actual layout of the program. It will only be WYSIWIS for users using the same front-ends or front-ends which interpret the document in the same manner. Documents which are not shared with other users may of course be stored in standard textfiles.

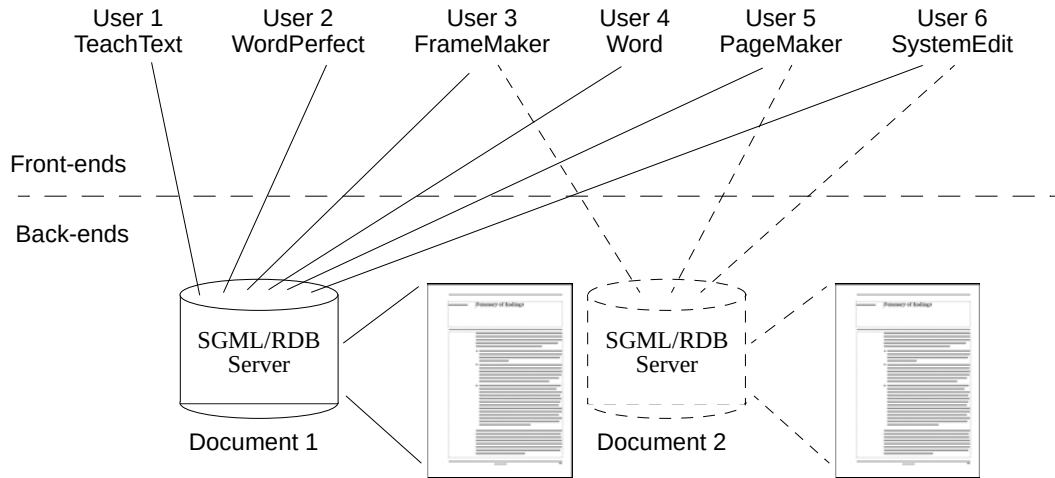


Figure 1: Separation of collaborative editing systems into front- and back-ends.

The Concurrency mechanism associated with transactions in databases is slower than other locking mechanisms and hinders highly coupled interaction requiring instantaneous updates. Most stages in collaborative writing do not require such highly coupled interaction and this does not seem to pose any problems. There is nothing inherent in the model which inhibits applications from making user actions visible across the system in support for user awareness. Change-bars, comments and annotations can easily be represented in the model.

We believe that any kind of collaborative editing will benefit from support of applications based on the SGML/RDB model. There would remain substantial problems for some types of collaboration requiring highly coupled interaction, but many problems have also been eliminated. In light of the fact that view sharing, floor control and telepointing are not supported by the model, it seems that the collaborative writing situations best supported by the model are the asynchronous writing situations.

The model is ideally suited toward writing documents describing facts, like manuals and documentations. Such tasks are common to collaborators combining information from and creating new documents for digital library networks [Fox 94, Bush 45]. In this type of writing the tasks can easily be divided between co-authors and the documents take advantage of the ability to use advanced Document Type Definitions (DTD). Co-authors in other situations may also be able to take advantage of the ability to use a variety of templates. We claim that using templates in collaborative writing can save time that would otherwise have been used rearranging and rewriting parts of the documents because the co-authors contributions become more homogenous.

Representation of Text in the Database

Representing data from a domain in a relational database most often reduces the redundancy of data. A correctly normalized data structure does not have any redundancies. Collections of documents vary often have redundant paragraphs of text. Representing text in a relational database gives us the same benefits as are found in other domains. Figure 2 shows an overview of the database structure which is motivated by several advantages.

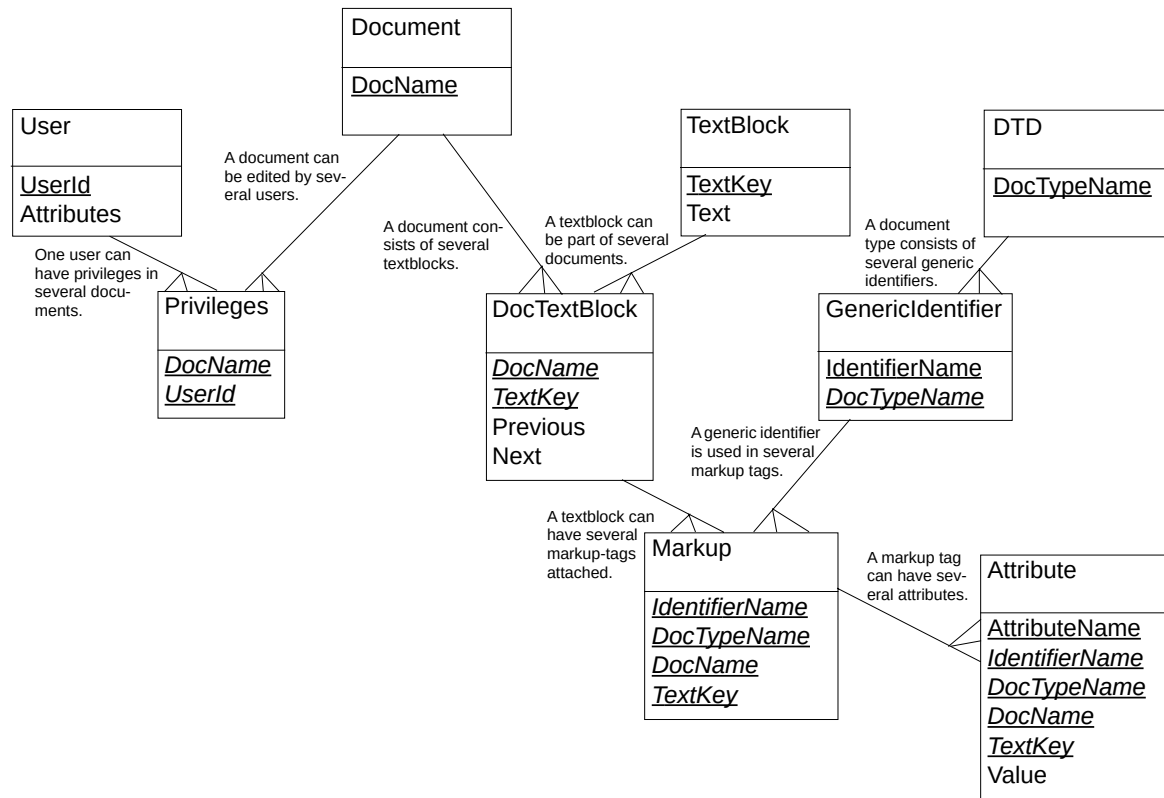


Figure 2: Structure diagram from the SGML/RDB model

The most important property is that several users may have privileges to edit the same document concurrently. The locking of text happens at the level of individual textblocks (records in the “TextBlock”-relation). A textblock could be anything from a character to a whole paragraph. It would be possible to define textblocks to be words, which would allow users to concurrently edit words adjacent to each other. The structure supports storage of one or more documents and the number of documents is only limited by the Database Management System (DBMS). Users are assigned privileges to edit documents in the “Privileges”-relation, which links users and documents. This relation also holds information about groups of co-authors collaborating on the same document. A document consists of a linked list of textblocks represented by the “DocTextBlock”-relation. The text related to a “DocTextBlock” record is found by a reference to the “TextBlock”-relation.

When more than one document is stored on the same DBMS it is possible to reference parts of other documents. A well known problem in larger organizations is that changes in one document has to be propagated to other documents and often no one knows exactly which documents needs to be changed. Instead of copying the pertinent sections the user can link (reference) the sections into his text and then apply his own markup. A feature like this would in our view be extremely useful. Whenever a change to the original sections was made, it would be updated (visible) in all documents including the sections. The user does not need to be aware which documents rely on which. This functionality is possible because we have separated text and the flow of documents

into two separate relations. A textblock belongs to all the documents that refers to it. In an actual implementation of this feature, the front-end should present the user with variations of this update. Sometimes users do not want documents to change without their control and the front-end system should support the storage of versions of the same document. If we call direct links between documents for *live links* we could call links that the user requests updates of for *update links*. When a user makes a update link the text of the other document is actually copied into his own document, but the reference is maintained in case the user requests a new update. This way the user could alter the referenced text without changing the original. Support for this functionality is not shown in the structure diagram.

SGML was meant as a language that permitted users to specify structures in documents with markup tags according to a Document Type Definition (DTD). The markup tags are embedded in the text. This implies that it is impossible to markup a text following more than one DTD. When storing text marked up with SGML in a database it is possible to separate the markup and the text in separate relations. This allows users to markup the same text with different DTD's. One might ask why this is an advantage? The relation between text and documents is not a one to one relation. In many situations documents reuse sections in other documents by referencing to each other. The documents do not necessarily follow the same DTD and our architecture allows users to markup a piece of text with the markup that applies to their documents. E.g. if a user wishes to submit a paper to several publications, where each publication uses different DTD's. It would not be necessary to make copies of the document and tag each individual copy. Yet another advantage of this approach is that following revisions to the text do not need to be copied to each document eliminating the risk of errors. The ability to associate several types of markup with a textblock is represented in the "Markup"-relation. Each markup tag in the text may have zero or more attributes. The "Attribute"-relation contains these links.

The document model of the SGML standard is system independent. It is not tuned to any particular software or hardware and all SGML-applications will be able to parse any document following the standard. This implies that an SGML-application may be at any level of sophistication. It may be very simple or very advanced. Element types (specified by markup tags) do not have to be known to the front-end application. Formatting for elements can be specified or redefined at any time. The user can also change between sets of processing procedures (formatting definitions) and view the document with different formatting without modifying the representation of the document.

Since the descriptive markup tells nothing about the way the text is formatted it is possible to design front-end applications which would support *private transformed views* [Shipman, McCall 94]. Private transformed views are individual sets of formatting procedures which can be associated with the same markup. A user could choose to view an outline as standard text, a tree or a bullet chart, and change between the alternate views.

This version of the model does not contain a representation of the actual DTD's related to the documents. We have not examined closely whether the DTD belongs in the model. The primary reason for not including it is that it is the responsibility of the front-ends to parse the documents and ensure that the documents follow the syntax of the DTD. The DBMS itself will never be able to check whether the syntax of the markup is correct. All that is needed is to have access to DTD's in either text files or as an extra attribute in the "DTD"-relation.

Discussion of Lessons Learned

We argued that we in our pragmatic approach will use “off the shelf” products to support collaboration. Our criteria for the products are:

- The applications must support the sharing of data.
- The applications must be cross platform compatible.
- The data representation must be a defacto industrial standard.

These criteria quickly narrow down our choices to different database management systems (DBMS).

The world of commercial applications consists primarily of programs based on heterogeneous datastructures and very few have the ability to support collaborative work on shared data. The available CSCW applications are domain specific and rely on proprietary models of collaboration. Multiuser database systems seems to be the only framework where users to some extent can collaborate on shared data. Due to the nature of databases the data stored in them is very often very standardized. Almost any company have a database with customers and possibly another one with suppliers. The data stored in the databases are simple relations of entities indexed by an identifier. Most of the operations on the data is very simple, and seldom reflect any sophisticated collaboration. Databases are designed to give the user the impression that he or she is the only user of the system. Concurrency control is maintained (provided) by the database management system (DBMS) and ensures that user operations do not collide.

Our use of database technology is quite different from ordinary use. We do not retrieve entities of homogeneous data but access parts of a larger more abstract datastructure. Users should be aware of the work of other users. Our use facilitates multiuser interfaces to multiuser backends and not singleuser interfaces to multiuser backends.

Client / Server Considerations

Since we have decided to use an “off the shelf” database server we will now examine what that decision implies for our model. Sharing data by a group of users can be accomplished in several ways. Without a client / server DBMS users only have single-user access to data entities via file-sharing. Multi-user access to entities with client / server database architectures vary in sophistication depending on how much work the clients must do. In some systems the clients do the actual searching resulting in a high continuous network load. For a client / server architecture to be affective the server must be a real engine that processes high level queries from clients and returns results. (E.g. SQL queries). Clients only issue queries and are free to support users in a responsive interface while they wait for results. In a collaborative environment updates in the shared data should be propagated to clients. Unfortunately database systems do not support notification about updates and clients must search for updates themselves in the shared data. An ideal “collaboration server” supports intelligent notification of clients about updates in the data relevant to each client. Clients could also install personal software “agents” on the server which would fire on patterns in the server that the user had specified. We then move away from the traditional database system toward a more object oriented architecture. This is a natural evolution: Using relational databases as “collaboration servers” is just a starting point for further cooperative development of our model.

Some text processing applications have solved the problem of multiuser access by supporting the merge of several documents when printing. This allowed groups of authors to work on the same document by storing different chapters or sections on a Work Group Server. Working with text in a database is similar to this concept. The document is still stored on an Work Group Server, but concurrent multiuser access is governed by the DBMS. In the traditional approach chapters corresponds to records in a database (although very large records). Under the SGML/RDB model the records are much smaller, which allows users to work within the same page just lines apart. One author can thus access another authors section (domain) for making references or annotations without interfering.

Why Relational Databases Using SQL?

One might ask why we investigate the possibilities of the relational databases at the time where object oriented databases are beginning to emerge. They are being build to support abstract datastructures and might better support collaboration on anything from text, graphics, sound and movies.

Our reasoning is again very pragmatic. Relational databases and the Structured Query Language (SQL) are defacto industrial standards. It is a matured technology available on any platform. Database technology took many iterations over three decades and relational databases slowly emerged.

The trend object oriented programming is relatively new one very few databases servers are commercially available. It will furthermore probably take another decade or more before the technology has matured and standardized database server is available on any platform.

We are looking for tools to support collaboration independently of platforms here an now. Object oriented architectures are not an option under these criteria and looking to develop our own collaboration support Server is not realistic either for several reasons:

- It would require a tremendous effort to design and implement.
- It would initially not be a platform independent solution.
- It would be hard to convince developers that they should support this idea.

There are several advantages to utilizing existing SQL client / server relational databases as a repositories for shared data:

- All issues of locking, concurrency control and data integrity are handled by the server. The model does thus not need to deal with these issues (SQL hides these).
- SQL has become a defacto standard as a query language in most relational database systems. The model is therefore application and platform independent.
- The interface is relatively simple and a non proprietary library will make it easy to take advantage of the model. It is there fore our hope that a critical mass of applications adhering to the model can be reached.
- The model supports the sharing of any structured data that can be marked up utilizing the Standard Generalized Markup Language (SGML)

The are also several disadvantages to using a SQL Client / Server architecture. The server does not support communication about other users work on the shared data. Two users may be working

next to each other without noticing anything before a collision of active workspaces happens. The ultimate scenario is a scene where users has a view or illustration showing each other workspaces. With a server that supported notification of clients about events on the server, client applications would be relieved from having to examine what changes has happened to the data. But there is no commonly accepted standard for how database servers notify their clients. Notification is furthermore also most often platform dependent which would make it difficult to work in a mixed environment.

Representation

Our work on the SGML/RDB model is still at its preliminary stages. We are still experimenting and simulating collaborative editing of technical documents in our current prototype. We have yet to demonstrate that reasonable performance can be achieved, and that the model gives sufficient support for collaboration on text. We hope to meet this goal in the near future by implementing new iterations of our collaborative editing system.

The granularity of text representation in the database is also another important issue. We want the model to be flexible enough to support various types of applications operating on the datastructure. Some applications would need to operate at the word or even at the character level and others would need to work at paragraph or page level. We must represent the data which is both efficient for most types of applications, but also accommodates representation needs of special applications.

It is also an open question whether document structures should be expressed implicitly or explicitly in the database. Even though front-end word processing applications have to parse the documents retrieved from the databases and build their own internal representation, it is still possible that some types of applications would require access to an explicit representation. E.g. applications that traverse hyperlinks or cross references. These data representation issues will also be addressed in our future iterations on our prototype.

Finally, we anticipate that the current work will be beneficial to developers and researchers who need an open interface to abstract datastructures. Because the data representation is separated from the application working on the data, we have the ability to let several applications operate on the same data independently of each other. The separate data representation forces us to employ a general representation which not application specific. In applications where the data representation is maintained internally, it is often difficult to add functionality to operate on the same structure. Furthermore if the “mother”-application that maintains the datastructure becomes obsolete then the “add on” applications will also become obsolete because they cannot themselves maintain the datastructure.

Conclusion

Our vision is a world in which users collaborate through applications of their own choice, sharing datastructures across operating system and hardware boundaries. Ideally, this vision would be enabled by a coordinated effort in the developer community to ensure that a standard for collaboration support is developed. Individual developers may try to develop more complicated models for collaboration support than the one outlined in this paper. We hypothesize that such models will not have great impact on collaboration between users in their domain, because the models will be proprietary and only support collaboration between users of the same program. Cross platform

collaboration will eventually be a reality. We argue that it is better to work toward this goal sooner rather than later.

As the path to standards and coordinated efforts are fraught with many obstacles, we argue that a pragmatic solution for today is to adapt “off the shelf” products. The client / server architecture we discussed is essentially a glue between users’ preferred front-ends and collaborative platforms. “Off the shelf” components lower the threshold to collaboration support by giving developers the tools to add the support at a low cost.

Bibliography

- [Beck, Bellotti 94] Beck, E. E., Bellotti, V. M. E., Informed Opportunism as Strategy: Supporting Coordination in Distributed Collaborative Writing. In Proceeding of the Third European Conference on Computer-Supported Cooperative Work (Milan, Italy, Sep. 13 - 17, 1993).
- [Bush 45] Bush, V., As We May Think. The Atlantic Monthly, Jul 1945.
- [Ellis et al. 94] Ellis, C. A. et al., Groupware: Some Issues and Experiences. In Communications of the ACM, Jan 1991 / Vol. 34, No. 1.
- [Fox 94] Fox, E. A., Images of Digital Libraries. Paper Version of Keynote Address for NORDINFO Conference: Digital transfer of images, Helsinki, Finland, Nov 1994.
- [Goldfarb 90] Goldfarb, C. F., The SGML Handbook. Clarendon Press, Oxford, 1990.
- [Grudin 94] Grudin, J., Eight Challenges for Developers. In Communications of the ACM, Jan 1994 / Vol. 37, No. 1.
- [McGuffin, Olson 92] McGuffin, L., Olson, G. M., ShrEdit: A Shared Electronic Workspace. Technical Report No. 45, Cognitive Science and Machine Intelligence Laboratory, The University of Michigan, 1992.
- [Neuwirth et al. 90] Neuwirth, C. M. et al., Issues in the Design of Computer Support for Co-authoring and Commenting. In Proceedings of the Conference on Computer-Supported Collaborative Work (Oct., 1990).
- [Olsen et al. 90] Olson, J. S. et al., Concurrent Editing: The Groups Interface. In Proceedings of the IFIP TC 13 Third International Conference on Human-Computer Interaction (Cambridge, U.K., 27-31 August, 1990).
- [Shipman, Marshall 93] Shipman, F.M., and Marshall, C.C. *Formality Considered Harmful: Experiences, Emerging Themes, and Directions*. Technical Report CU-CS-648-93, Department of Computer Science, University of Colorado, Boulder, 1993.
- [Shipman, McCall 94] Shipman, F.M., and McCall, R. Supporting Knowledge-Base Evolution with Incremental Formalization. To appear in *Proceedings of CHI'94* (Boston, Mass., Apr. 24-28). 1994.