Towards a new approach to tightly coupled document collaboration

Speculative Short Paper

Stijn Dekeyser

Mathematics and Computing
University of Southern Queensland
Queensland 4350 Australia

dekeyser@usq.edu.au

Abstract *Currently document collaboration typically proceeds using tools such as* CVS *or vendor-specific Computer Supported Collaborative Work (*CSCW*) and Electronic Meeting (*EM*) messaging systems. Both regulate essentially asynchronous* loosely coupled *collaboration. The prime disadvantages of these technologies are that often documents are checked out or distributed in their entirety and that human interaction is needed in case of unresolvable conflicts. On the side of* tightly coupled *distributed collaborative work, emerging XML databases are employing database-type concurrency control techniques, but unfortunately tend to lock entire documents preventing simultaneous updates. XML-enabled relational databases have the same intrinsic problems, leading to the question if another way is possible.*

In this speculative short paper we describe a novel approach toward tightly coupled document collaboration, involving database-style synchronous client-server collaboration tailored to semi-structured documents. It is partly based on previous theoretic results which introduced path locks to control concurrency on semi-structured data. We also describe how clients may use a future communication protocol based on the path locks.

Keywords Document and XML Databases, Document Management, Document Collaboration.

1 Introduction

Collaboration on digital documents is not only a hot topic, it's a very important one as well. The economic benefits of software allowing easy collaboration on any kind of digital information are enormous. Currently there is a wide range of techniques available, both for general and specific purposes. Often these techniques are classified using the two dimensions of *time* and *location*. Two main classes are physically distributed *asynchronous* collaborative work, and physically distributed *synchronous* collaborative work (we do not discuss collaborative work in one location). We list some of them here, and give a very brief indication of some of their individual drawbacks. Then we proceed with outlining a new approach to document collaboration.

Proceedings of the 9th Australasian Document Computing Symposium, Melbourne, Australia, December 13, 2004.

Asynchronous, loosely coupled

CVS and related technologies. **Drawbacks:** (1) The update method is based on lines in text files, rather than on a conceptual representation of the semantics of a document. (2) In case of multiple updates on a given line, the system requires human intervention to solve the conflict.

Tracking The technique of marking-up sections of a document with change information is typically used by word processing software and other office tools. **Drawbacks:** (1) Multiple instances of a document exist among authors, making file management by hand unavoidable. (2) human intervention is needed to solve conflicts. (3) Documents must be distributed via email, disk, FTP or other means.

CSCW and EMS (Messaging) A large body of commercial systems manage group work from a technical, managerial, and social perspective [10, 16, 1]. The level of automatic synchronization differs from product to product, but often the entire workspace is distributed at each participant's site, while each copy is kept up-to-date by interchanging appropriate control messages [12]. **Drawbacks:** Apart from those of the previous two technologies, a distributed groupware system can suffer concurrency control problems due to events arriving out of order. Other problems are outlined in [12].

Synchronous, tightly coupled

XML-enabled DB Documents can be stored in relational tables, and users can update them using transactions which are based on classic concurrency control mechanisms. **Drawbacks:** it has been shown [4, 9] that relational databases, also those that are XML-enabled, are far from adequate for generic document collaboration

purposes. This is mainly due to locking mechanisms being table-based while several parts of a document (or its entirety) may be stored in the same table.

Native XML DB A document may be converted to XML and stored in emerging native XML databases [4]. **Drawbacks:** as with relational databases, locking mechanisms are currently primarily document-based, meaning that concurrent updates on a single document are not allowed.

Looking only at synchronous collaboration systems, we notice that the main drawbacks of current methods are related to the unsuitability of the concurrency control technique that is being used. What is needed is a new approach that controls concurrency at a logical level within semi-structured documents (e.g. XML [17]), rather than on a document-per-document basis. We will present such an approach in Section 3.

2 Usage Scenarios

In this section we briefly describe some application areas where a new type of collaboration would be beneficial.

2.1 Writing Documents

Typically, researchers using LaTeX collaborate on papers with co-authors using CVS or sending fragments to a designated editor. Authors using other word processors for more general purposes typically exchange entire documents, with parts marked up to represent changes made by different persons. This method is even less satisfactory than using files in CVS repositories.

An alternative approach where the document is stored on a server and clients interact with it synchronously allows for a far better collaboration experience. If a semi-structured locking mechanism is used by the server, logical sections of a document, however small or large, can be locked for write-access. This ensures that the semantics of the document is used in regulating concurrent access, and that human intervention is not required to solve conflicts. Such an approach would also make sense for web content management systems.

2.2 Drawing Plans

Architects, engineers, or artists working in a team on a vector graphic currently have even fewer options than authors of text-based documents [5, 6, 8], unless the graphic can be saved in a text format and kept in a CVS repository as described above. Certain products have facilities akin to those of word processors, where parts are marked up to indicate editing by different persons. However, someone must still manually manage a master document and decide which updates are retained. Other products use collaboration mechanisms based on messaging. Here, too, multiple instances of a single drawing exist, and users must intervene in the management and resolution of conflicting updates. Certainly, in some cases these properties are beneficial, but more often they are not wanted.

Here, too, a different approach would likely be better suited. Consider that the graphic is stored as SVG [11] on a document collaboration server that uses the document's semantics to allow users concurrent access to logical parts of the document while scheduling potentially conflicting updates. Collaborators would see the entire graphic evolve as changes are made to it. Individual authors could be given read- and update restrictions on any logical part, akin to database privileges on tables or views. Only one consistent, authoritative instance — managed by the document collaboration server — exists (but versioning is possible as in Section 2.3) at any given time, although individuals may be allowed to make copies of any version branch.

Some of these features are offered by DR. DWG [6], but this is not a generic tool for use outside the particular application. Furthermore, it involves storing the entire drawing at each user, letting one user at a time make changes which are sent to the other collaborators in real time. Similar comments hold for [13].

2.3 Programming

As a final use case, consider how collaborative programming could be vastly improved if program source code is treated logically rather than as a sequence of lines. As in Section 2.1, the document collaboration server regulates concurrent updates from different clients, disallowing conflicts. The important advantage of using CVS, not just controlling updates but also managing versions, can be effectively simulated in this approach as well. For this, consider that the source code is represented as an XML tree [3]. Special tags at any location in the tree can indicate different versions of code. A smart editor (e.g. Epic [2] already does this) may manipulate these elements in diverse manners, making version management realistic and flexible.

3 Path Lock Concurrency Control

In previous work [9] we have presented a novel concurrency control technique for use in schedulers for semi-structured databases. It uses a new kind of locks which are closely related to XPath queries to indicate precisely which logical parts of a document have been read or written by individual clients.

Clients *query* a document by sending XPath-like statements to the document server. Appropriate path (read) locks are set in the document. The document server's scheduler checks for conflicts with other clients' locks and sends back a node-set if there are no conflicts.

Clients may also *update* parts that they have queried within the context of a transaction, resulting in addi-

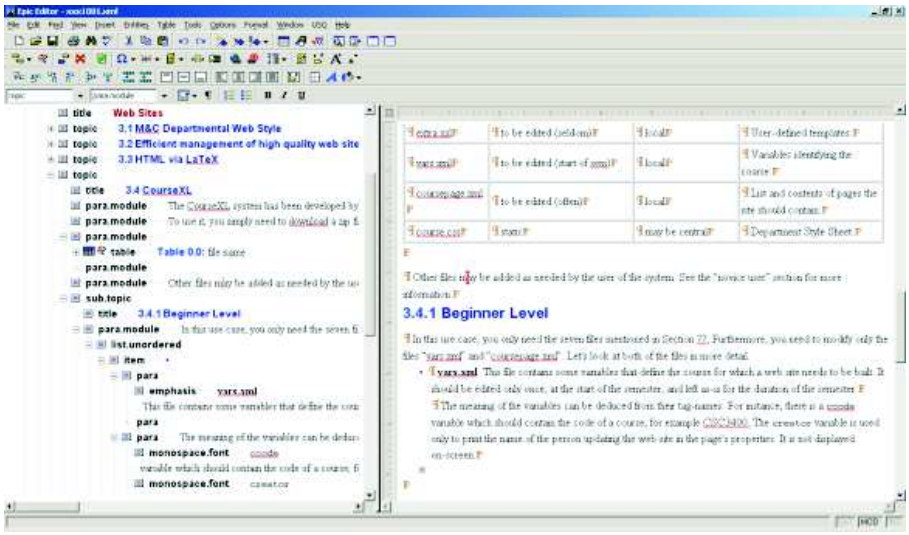

Figure 1: Epic Client View

tional path (write) locks. Path lock conflict rules are described which enable checking for conflicts. The conflict rules exist in two versions, *Path Lock Propagation* and *Path Lock Satisfiability*, representing a trade-off between time and space complexity in conflict checking algorithms using the two schemes.

The paper also introduces two schedulers, a *Conflict* Scheduler and a *Commit* Scheduler. Both schedulers guarantee serializability of schedules consisting of query and update statement belonging to various transactions. Furthermore, their use of Path Locks and corresponding conflict rules allow for a much higher level of concurrency compared to traditional relational and hierarchical methods [9].

3.1 Transaction Operations

Making the above more concrete, we briefly list the operations that clients may use within transactions. Some details, notably constraints on their use, are omitted to increase readability.

A(n, a) The *addition* operation creates a new node n' with tag-name a and an edge (n, n') in the document tree.

D(n) The *deletion* operation removes the node n and the edge incident to n in the document tree.

Q(n, p) The *query* operation returns the set of nodes selected by the path expression p started from the context node n.

C$()$ The *commit* operation signals the end of a transaction.

Note that the above update operations are sufficient to support an XML update language such as XUpdate [14], and that they can update both content and structure.

4 Communication Protocol

As yet unexplored in the context of this paper is a communication protocol (DCP — *document collaboration protocol*) for use by document collaboration servers and client document editors. Clearly, commands sent by the client include the operations listed in Section 3.1. Additionally, a *rollback* and *abort* operation will be needed, as is a *handshake* operation that identifies client and transaction to the scheduler.

Replies from the server include sets of nodes as a result for queries and updates, and notification of conflicts, and failures of both operations and transactions.

As a protocol for communication between a database server and client, it is apparent that state must be preserved during the lifetime of a transaction. And since this technology must be usable over the Internet, integration of the protocol with IP is essential.

5 Client Issues

Perhaps the most exciting challenge to realize this new approach to document collaboration lies with redesigning graphical clients. A general-purpose XML authoring client such as Epic offers a flat and a hierarchical view of the document being edited (see Figure 1).

Clients implementing the DCP protocol described in Section 4 must show the contents of parts of the document as being uncertain. When a client (say, `A`) first starts talking to the server, the full document is received, yielding a flat and hierarchical view as in Figure 1. However, to allow others to edit the document, client `A` must *commit* to release its read locks, which at that point cover the entire document. An end-user working with `A` can subsequently traverse the *assumed* document (possibly requesting refreshes whenever needed), while querying a specific part of the document immediately prior to making a change.

Clearly, the exact query posed before the update will determine the level of concurrent authoring allowed by other clients. Thus, client `A` must allow for an intelligent way to describe the section that will be modified. Many different queries are possible; for example, a positional query (`/section[3]`) may be preferred over a content-specific query (`/section[title='ab']`), with both yielding different path locks and hence resulting in different concurrent updates being allowed or refused.

The above can be summarized as the following formal research problem: given a set of path locks L and a write operation o, give all query expressions q such that the locks required by o and q do not conflict with L.

In addition to intelligently querying the part of the document that the end-user is editing, the client must be flexible in displaying concurrent changes (the conflict rules make sure that no problems can arise, however). For instance, if `A` is working on a specific section while `B` is moving that section to a different location, then upon `B`'s commit, `A` should refresh its view to show the new location of the section being edited. These and other issues with clients need to be researched for both general-purpose and specific-purpose (e.g. drawing) editors.

6 Realization

To realize this type of document collaboration, at least five areas need further research and/or development:

- The Path Locks technique referred to in Section 3 needs to be expanded to allow for more expressive update commands and to support queries expressed in the full surface-syntax of XPath [7]. This research is proceeding smoothly.
- The communication protocol described in Section 4 needs to be defined exactly by network protocol experts. Integration with the Internet Protocol is essential.
- A general-purpose document collaboration server needs to be extended with an implementation of the Path Locks concurrency control theory and the communications protocol. An open source native XML database server such as eXist [15] would be a good target.
- As an alternative to a general-purpose document collaboration server, existing applications such as word processors, vector graphics authoring systems, spreadsheet programs and others could to be extended to include a light-weight server allowing others to collaborate using the communication protocols in their clients.
- Clients need to be re-thought to allow parts of the document they are editing to go out of date, and read-lock only those parts that are actively seen by the end-user, and write-lock the even smaller fragment that is being updated at any given time. Research is necessary to investigate how this can be done generally, while allowing a maximal level of concurrent updates.

References

[1] *Proceeding on the ACM 2002 Conference on Computer Supported Cooperative Work*, New Orleans, Louisiana, USA, November 2002. ACM.

[2] Arbortext. Epic editor overview. Arbortext.com Web Article, 2004. `http://www.arbortext.com/html/epic_editor_overview.html`.

[3] G. Badros. JavaML: A markup language for Java source code. In *Proceedings of the Ninth International Conference on the World Wide Web*, May 2000.

[4] R. Bourret. XML and databases. Website, 2004. `www.rpbourret.com/xml/XMLAndDatabases.htm`.

[5] B. Burchard. Sharing your drawings: An introduction to collaboration. Autodesk.com Web Article, 2004.

[6] Dr DWG Knowledge Center. The CAD View Collaborator. White Paper, 2004. `http://www.drdwg.com/techcenter/cadviewcollaborator.html`.

[7] J. Clark and S. DeRose. XML Path Language (XPath). Recommendation, World Wide Web Consortium (W3C), 1999. `http://www.w3.org/TR/xpath`.

[8] P. Coffee. Collaboration's new age. eWeek.com Web Article, February 2004. `http://www.eweek.com/article2/0%2C1759%2C1539706%2C00.asp`.

[9] S. Dekeyser, J. Hidders and J. Paredaens. A transaction model for XML databases. *World Wide Web Journal*, 2004.

[10] D. Eseryel, R. Ganesan and G. S. Edmonds. Review of computer-supported collaborative work systems. *Educational Technology & Society*, Volume 5, Number 2, 2002.

[11] J. Ferraiolo, J. Fujisawa and D. Jackson. Scalable vector graphics (SVG) 1.1. Recommendation, World Wide Web Consortium (W3C), January 2003. `http://www.w3.org/TR/SVG/`.

[12] S. Greenberg and D. Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In *Proceedings of the ACM conference on CSCW*, pages 207–217. ACM Press, 1994.

[13] C. Ignat and M. Norrie. Grouping/ungrouping in graphical collaborative editing systems. In *ECSCW*, 2003.

[14] L. Martin. XUpdate – XML Update Language. Draft requirements, XML:DB, November 2000.

[15] W. Meier. eXist: An open source native XML database. In *Web, Web-Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops*. Springer LNCS Series 2593, 2002.

[16] J. F. Nunamaker, Alan R. Dennis, Joseph S. Valacich, Douglas Vogel and Joey F. George. Electronic meeting systems. *Commun. ACM*, Volume 34, Number 7, pages 40–61, 1991.

[17] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler. Extensible markup language (XML) 1.0. Recommendation, World Wide Web Consortium (W3C), February 2004. `http://www.w3.org/XML/`.