Transaction Management in Collaborative Virtual Environments

Jack J. Yang, Gail E. Kaiser and Stephen E. Dossick
Computer Science Department,
Columbia University,
New York, NY 10025
{jyang, kaiser, sdossick}@cs.columbia.edu

Abstract

Collaborative Virtual Environments (CVE) are groupware systems in which data are represented as objects in a virtual world. CVE supports activities ranging from software design and development to health care management where user operations are long in duration and intensive in human interaction. We found the advanced database transaction modeling and management technologies that has developed in the database community very useful in the design and development of CVE. In this paper, we first present a dynamically customizable transaction manager, JPern, then describe how a CVE could benefit from integrating with such a transaction management service. Finally, we describe an instance of CVE that we developed, CHIME, and how to integrate JPern with it.

1 Introduction

Collaborative Virtual Environments (CVE) are groupware systems that provide a visual context for multiple users, supporting a wide range of activities such as software design and development, scientific data browsing, and health care management. In these systems, various kinds of information is usually presented to the users as objects in a virtual world, and the data operations are represented by the users' action against these virtual objects. Examples of CVE can be found in [6, 8]. The design and implementation of CVE is still an active research topic [1]. Current CVE design research is very heavy on user interface issues such as data modeling and visualization.

Transaction management is a rather traditional technology in Database systems [3]. Transactions are grouped data operations that are executed with certain consistency guarantees. Originally, a transaction is atomic (operations are executed in an all-or-none fashion) and isolated with each other. Later, a number of Extended Transaction Models (ETM) [7] have been proposed for the kind of human-oriented, long-duration activities frequently seen in CVE. These ETM provides advanced and flexible correctness criteria for the transactions, and allows more non-isolated activities among the users.

By its nature, Virtual Collaborative Environments involves multi-user activities, and these activities are expected to be collaborative, very much in a way that the original atomic transaction models would not accommodate and the ETMs are more suitable. When a CVE is constructed, it could be built from scratch with specific requirements of the environment, making its own database, user interface and other components. Consequently, the correctness criteria of the data management can be hardwired into the system. On the other hand, the component-based approach, with the adoption of underlining components such as data repository, concurrency control, caching and UI, allows a CVE to be constructed with less effort.

With the development of the World Wide Web (WWW), the component-based approach is becoming more and more popular. Also, when a CVE intends to work with data sources such as web servers, one might not even have the luxury to customize the data sources. Even if one wants to custom build a CVE, (s)he must live with the fact that there are pre-existing components that cannot be customized as (s)he wants.

In the recent years, component-based transaction modeling and management technology advances have made the realization of ETM easier and flexible [9, 12, 13, **Error! Reference source not found.**]. We are particularly interested in the application of such technology in the design and implementation of CVE. In this paper, we first briefly examine how JPern, a flexible transaction service for internet-based collaborative systems, works, then describe in general how CVE could benefit from such transaction technology. Finally, we present a concrete example of how to integrate a CVE with a transaction manager to obtain such benefits.

2 JPern: Flexible Transaction Service

In [**Error! Reference source not found.**], we presented a novel transaction server architecture for internet-based, collaborative systems. Here, we need to briefly go over what it offers in terms of application to the CVE construction.

The core of the architecture is a transaction server that operates independently of the data sources and applications. The transaction server directly accesses the data sources, therefore it has the opportunity to leverage possible Concurrency Control (CC) features provided by the data sources. The transaction server can also be customized by the applications to realize various ETMs. The customization mechanisms include changing the lock table, writing CC policies callbacks, and writing data access modules to utilize the CC features provided by the data sources.

2.1 Middleware Approach

There have been attempts to build CC capabilities such as locking, check-out/check-in, and even distributed transactions into the WWW infrastructure. There are two major approaches, categorized according to where the bulk of the functionality is placed, the server side or the client side. The server approach [14] introduces CC functionality into the web servers so that the clients can explicitly submit operations as part of a transaction; a collection of cooperating web servers can then realize distributed transactions. The client approach [5, 11] places CC policies into the clients, but still often requires the servers to provide very basic CC building blocks such as locks or versions.

Neither of these approaches addresses the problem of how to dynamically customize the ETM available to the applications. Hardwiring CC policies into either the data sources or the applications greatly limits scalability and flexibility.

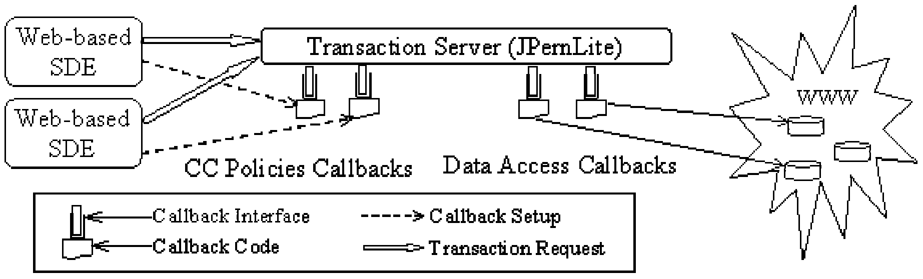

Figure 1 Transaction Server Architecture

Figure 1 describes our proposal of a third, middleware approach: the CC policies are realized in a new component, the external transaction server. The SDEs make requests to the transaction server, which carries out the concurrency control policies in the process of obtaining the data from the web servers on behalf of the applications. The web servers are not required to provide any CC support at all, although any support they do happen to provide can also be exploited. The major advantages of this approach are:

- The transaction server can easily be tailored to implement the desired ETM.
- The approach does not require any changes to the servers, or the applications, although it does not prevent the server from being integrated at source code level.
- Coordination among applications that share data can be done in the transaction server and resolving the difference between the ETMs used by these SDEs is made possible.

We have a proof of concept implementation of the transaction server, called JPern. JPern can be treated as a normal web server where special URLs are treated as transaction requests. Figure 2 shows sample request and reply messages between a potential application and JPern.

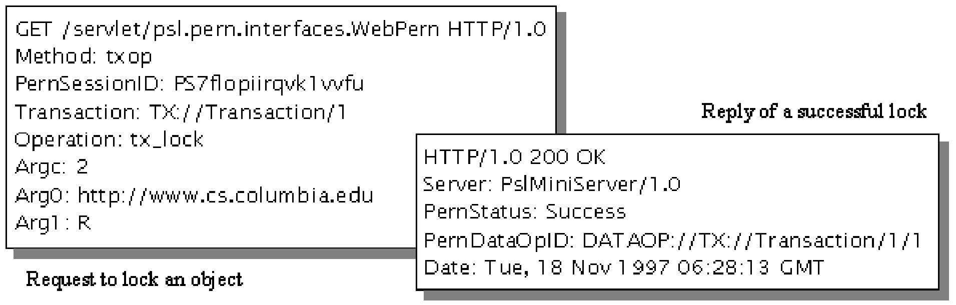
```
GET /servlet/psl.pern.interfaces.WebPern HTTP/1.0
Method: txop
PernSessionID: PS7flopiirqvk1vvfu
Transaction: TX://Transaction/1
Operation: tx_lock
Argc: 2
Arg0: http://www.cs.columbia.edu
Arg1: R
```
Request to lock an object
Reply of a successful lock
```
HTTP/1.0 200 OK
Server: PslMiniServer/1.0
PernStatus: Success
PernDataOpID: DATAOP://TX://Transaction/1/1
Date: Tue, 18 Nov 1997 06:28:13 GMT
```

Figure 2 Sample Communication between JPern and an application

The request is to lock an object whose URL is `http://www.cs.columbia.edu` in Read-only (R) mode. The values of `PernSessionID` and `Transaction` ID were previously obtained from JPern. The reply message shows the successful execution of such a request, and the message tells the SDE that the object has been locked.

2.2 Customizing JPern

The CC policies of JPern can be customized to implement the ETMs that the applications desire. Here we can only briefly go over the areas where one can customize JPern. For more information, see [**Error! Reference source not found.**].

- Lock Table: JPern uses a lock compatibility table to describe the different lock modes used by the ETM it implements.
- Lock Semantics: The fine-grained semantics of each lock mode is defined in the callback code that the administrators of each application write. Such callback code can be transferred to JPern dynamically.
- Transaction Modeling: The ETMs are implemented by a combination of customizing the lock table and writing the callback code.
- Exploiting CC Features provided by the Web Servers: JPern also has a number of callbacks that the application administrator can set up to access the web servers. Such callbacks can be used to allow JPern to use the CC features provided by the web servers, such as locking a web page, starting a two-phase commit, or spawning a new version of the web page.

3 CVE and Transaction Management

As we mentioned in the introduction, one might build a CVE from scratch or by using existing components. But no matter which approach is being taken, we can almost always find the following logically separable components in it:

- One or multiple *data sources* that store the shared data. The CVEs may have geographically distributed data sources, and in case of web-based CVEs, the web servers become part of the data sources.
- A graphical *user interface* that visually represents the data. As CVEs are strong in data representation and 3-D modeling, their user interfaces are almost certainly full of animations and other computation-heavy operations. Therefore, how to accelerate the UI to achieve a near to real-time response is also an important issue.
- A *groupware engine* that embodies various policies and functions of the CVE. Because the CVEs are supposed to support certain productive activities such as a design process, a groupware engine that supports workflow at a certain level is very important. Some CVE may only have strong data visualization and very "dumb" groupware engine, but in general, a CVE can be expected to contain such a functional unit.

For example, in [8], the Maverick system consists of the Spatial Management System (data source), a Graphics System (UI) and a number of groupware engine units such as the Constraint-based Manipulation Collision Detection and Navigation subsystems.

A transaction manager such as JPern could be deployed in all of the above components. In this section, we will look at how we could apply JPern in these components.

3.1 Data Management

The most traditional and natural application of transaction is in data management. In CVE, multiple users access distributed data sources in the same time, and the concurrency control problem becomes an immediate issue.

When the data sources are custom-built, the concurrency control functionality and policies can be specifically coded into them. However, the data sources may be pre-existing databases or web servers that provide a certain fixed form of concurrency control. The CC policies enforced by the data sources may not be suitable for the specific CVE. Depending on the level of support the data sources have, the CVE needs to map the desired transaction model differently:

- Data sources with no concurrency control support: the whole transaction model would be written in plug-in code, and run solely in JPern. As we have mentioned before, the applications that do not go through JPern will not be protected against inconsistencies such as dirty reads or broken links.
- Data sources with locking capability: using the locking capability in the data sources, we can ensure the applications that do not go through JPern or DIS will not get any dirty reads.
- Data sources with distributed transaction notions: such data sources include web servers that implement TIP, or the various HTTP-wrapped traditional databases. For these data sources, JPern not only can be used to guard other applications from dirty reads, but also allow distributed commit (with 2-phase commit) and delegate sub transactions to the data source. These data sources mostly implement atomic transaction models for the transactions running solely within their scope. When multiple data objects are accessed in one data source, we can conveniently utilize the transactions in such data sources.
- Data sources with extended transaction support: not only can we use the transactions to ensure the atomicity of multiple data access in one data source, we can also make use of the extensions they provide to ease the writing of the transaction plug-in code in JPern. However, a mapping between the desired transaction model and the transaction model provided by the data sources is not always easy to find.

In all of these cases, JPern stands in between of the distributed data sources and coordinates the data sources and the rest of the CVE. The CVE configures the CC policies in the JPern by the means we mentioned in Section 2.2, and when there are multiple CVEs sharing the same set of data sources (e.g., web servers), they must go through the same JPern instance in order to be fully coordinated.

3.2 User Interface

In CVE, there are issues concerning the UI update speed and the data processing speed at the back end databases. A number of cache algorithms concerning the faster display of virtual world changes have been devised. Most cache algorithms use optimistic concurrency control policies: when an operation is done, it is assumed that there will be no CC conflict, and the operation is carried out in the local cache. In the meanwhile or a bit later, the backend data source is also contacted to perform the real operation. If the operation turns out to be conflicting with other on-going transactions, it will be rejected and the local cache needs to be rolled back. For example, assume the explosion of a ball in a certain theme represents the deletion of a file. When a user performs a delete operation, the UI module deletes it from its local cache, start displaying the explosion animation, and in the meanwhile, contacts the backend data source for a real deletion. However, the deletion might be rejected by the VEM because of concurrency control limits. Therefore the UI might decide to “undo” the effect of the explosion. Depending on the timing, the undo might be a back-to-the-time fashion undo of the explosion animation, or simply a regeneration of the ball. In order to support such application specific requirements, JPern can be used to manage the local cache.

3.3 Groupware Engine

When the UI updates the virtual world (i.e., view of the data objects), it may be desired that only a portion of the world should be updated. For example, imagine we have an automobile design environment where, among other designers, two designers A and B work closely. A is the expert on engine design, and B works on the transmission. When B has a new idea about the transmission and wants to see how it fits the designs of the engine, he might want A to see his new design, put together with A's engine design and run a few tests. In that case, B's design activity, which is encapsulated by a transaction, is made visible to A. However, the design is not committed to the center database, and any other designer will not be able to see that. There might be other users who currently have the transmission design document in their view, but what they see is still the old version until B (or A, if the transaction model allows a co-author to commit the transaction) commits.

The groupware engine takes the operation requests from the users, and before it issues update or access requests to the data sources, it goes through the necessary steps to finish its own task such as the workflow that is being executed. The concurrency control or transaction management here is at a higher level than that in the data sources. As in the above example, the groupware engine concerns more about whether or not the users should “see” certain data changes and when the see it according to the application specification of the CVE. In data sources, however, concurrency control focuses more on the data consistency, either in a single site or a distributed one.

Finally, let us look at the "view update" issue. When data are updated in long-duration, and often advanced, transactions in the databases, the user interfaces need to be updated in a similar way too. In traditional database systems, users or applications actively update and query the database. The changes made by one user are not actually "seen" by others until they make new queries. The correctness of the queries is ensured by the assumption that the queries are independent and executed in separate transactions. However, in a virtual environment where the whole database (or databases, in a distributed situation) is visualized as a 3D (or sometimes 2D) objects, and the users make certain assumptions about the virtual world they see, especially on how it changes regarding other users' update requests. One common assumption, which is used widely in the World Wide Web by the browsers, is that a user's view never changes until he or she clicks the "refresh" button. Reflected to CVE, the assumption suggests that the display of the virtual world is always a frozen snapshot of the databases. The snapshot can only be updated when the user requests it to. Users navigate and operate on the frozen version of the world may meet problems such as trying to update an object that has been deleted or locked in a conflicting lock mode by other users. Because the virtual world appears to be a "reality" to the users, this kind of unforeseeable failure frequently annoys the users.

An alternative is that the system tries to keep the virtual world up-to-date. That is, when someone changes an object in the databases, the system actively updates the display of all the users who are affected. The “push” method better synchronizes the users’ views, but it generates problems such as “whether or not to update a user’s view if the data change has not been committed yet”? Furthermore, different users may decide to “see” the different transactions, or different versions of the database. Therefore, it would be desirable to have query interface for the ongoing transactions. JPern provides both a human readable HTML interface and a machine-readable HTTP header interface to query existing transactions. Not only the ongoing transactions can be queried, but also the committed and aborted transactions that haven’t been cleaned up by the CVE can also be queried. Therefore the CVE users can “replay” past events in the unit of transactions.

4 CHIME

CHIME (the Columbia Hypermedia Immersion Environment) [6] is a CVE framework that aims to synthesize results from both the User Interface research community and the Hypermedia research community. At one hand, the recent years of User Interface research work paid much attention to the development to the techniques to help users assimilate a broad range of related information easily, through the development of 3D-based information visualizations, such as [4]. These techniques help visualize a large amount of related yet heterogeneous information in the 3D world, making it easier to be digested by the users. On the other hand, the Hypermedia research activities have produced tools and systems that automatically "link" documents, not only at object/document level, but also at a finer granularity such as an item or a paragraph in a document.

CHIME targets on the creation of systems that supports the wide range of collaborative activities, especially in software engineering related areas. Software engineering activities typically involve more than coding and design: from requirement analysis, cost analysis, to test design, code review reports, schedule management, to user survey, bug reports, there are a big variety of documents and activities involved.

CHIME supports such a wide variety of in a XML-based meta-data architecture. It has three layers of data representations. The Data Integration Server (DIS) keeps meta-information about the raw, real-world data from potentially multiple data sources. For example, source code can be retrieved from a configuration management system [15], design documents can be found in a document management system such as BSCW [2], emails, discussions, decision making polls can be found in online groupware tools such as eRooms [10]. DIS does not store the actual data, but rather meta-information about them, such as author, permissions, creation date, modification history, etc. The data is organized into a multi-root tree of XML elements (Xanth dataElement), and DIS maintains an XML document that describes the whole set of data.

The Virtual Environment Model (VEM) server maps real-world data to one of the four types of virtual objects: Container, Component, Connector and Behavior. Using an analogy of the file system, "Container" objects are like directories of folders that may contain "Component" objects like files. The "Connector" objects serve as links that connects two or more Containers. "Behavior" objects describe the kind of operations that one can do to a certain object. The VEM layer distills the various real world data type to a much simpler model, and therefore the mapping between the data and representation could be made easier and in a more standardized fashion.

Finally, a Theme manager maps the VEM objects to user-understandable 3D objects in an application-specific fashion (called Theme). In CHIME, the users interact with the real-world objects by navigating in the 3D-world presented by the UI. Each Theme contains a set of plug-in code that can be run in the CHIME Client to represent the objects in the proper style according to the role of the user and the type of the client.

For example, an HTML page would have its URL and various meta-information such as creation time, last-modified time stored in DIS, and it would be represented as a component object in VEM carrying the meta-information of the page as attributes. Finally after the Theme manager applies a specific Theme to the CHIME Client, the HTML page might be displayed as a chair in a room (if the Theme consists of furnitures and rooms) or a tree on a planet (if the Theme consists of trees and planets).

Figure 3 shows the general architecture:

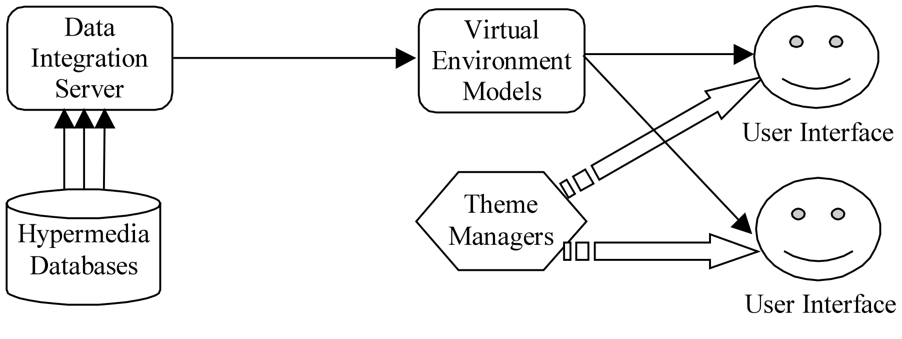

Figure 3 CHIME Architecture

Transaction Services are needed in CHIME, in multiple forms and places. We will discuss the various alternatives and utilities in the following sections.

4.1.1 JPern for Data Management

The list of Transactions and their contents are exposed to the UI by JPern via DIS and VEM. JPern exposes all the transactions that are not yet cleaned up via a human readable HTML interface. It can also export the transaction activities through XML, which is more machineoriented. The UI can then process the list of transactions, and provides the user with the list of transactions that can be updated to the virtual world. In our example case, A would choose the transaction that contains B's transmission design and update the changes in his/her UI component.

4.1.2 JPern In User Interface

When the user performs any data changing operations, the UI execute that in a UI-Transaction, which is defined in JPern as a sub-class of the Transaction class. The UI-Transaction has a error_lock method defined which calls back to the Graphics Engine to stop the current animation and start showing the compensating animation (e.g., the reappearance of the ball).

The Cache Event Handler as shown in Figure 4 may also update other users' "view". Consider two users co-authoring on a set of design documents. They, according to their collaboration model, may share intermediate editing results. Normally a user reveals changes to the virtual world by requesting a "refresh" at UI, which re-fetches the VEM objects and re-draw the 3D world. But in this case, the updates of the local UI transaction of one user need to be reflected to the co-authors' UI in a more active fashion, similar to the concept of the "push" technology.

When a user logs on, the CHIME Client searches for his/her co-authors, and adds this new user to the co-author's transaction as an additional "watcher". Then, when the co-author updates an object in the transaction, the before_update method defined by the UI transaction looks up the "watchers" and sends update messages to their CHIME Client as well.

Figure 4 depicts the interactions between JPern and the components of UI component. JPern serves as a local cache manager in this case, managing the local locking of the VEM objects. The Graphics Engine also registers its drawing routines with JPern as transaction plug-in code.

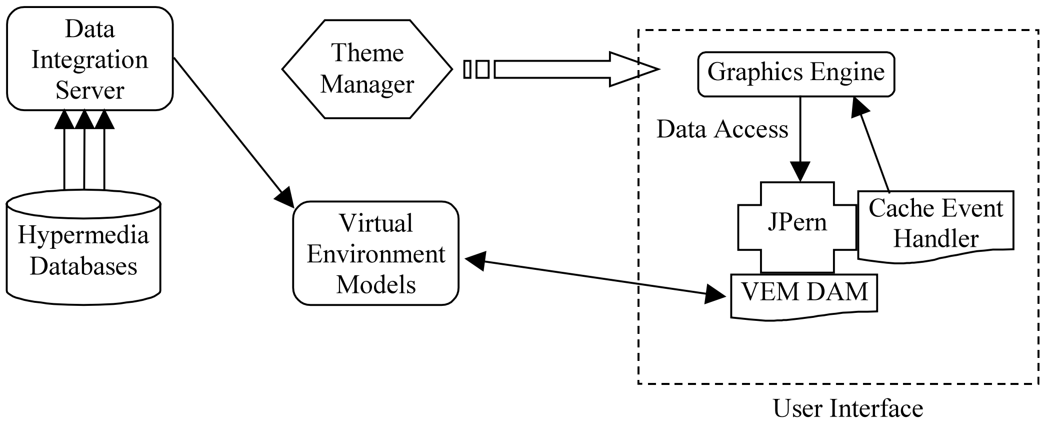

Figure 4 JPern in User Interface

The advantage of using a transaction manager to handle the UI issues is two-fold. First, it conceptually separates the UI issues in a collaborative system into the graphical representation issues and consistency maintenance issues, breaking down big problems into smaller pieces.

Second, from the software system's point of view, by using a transaction manager to handle the concurrency control policies, crash recovery and rollback policies, we get a more componentizable thus more maintainable software system architecture.

4.1.3 Collaboration Among Users

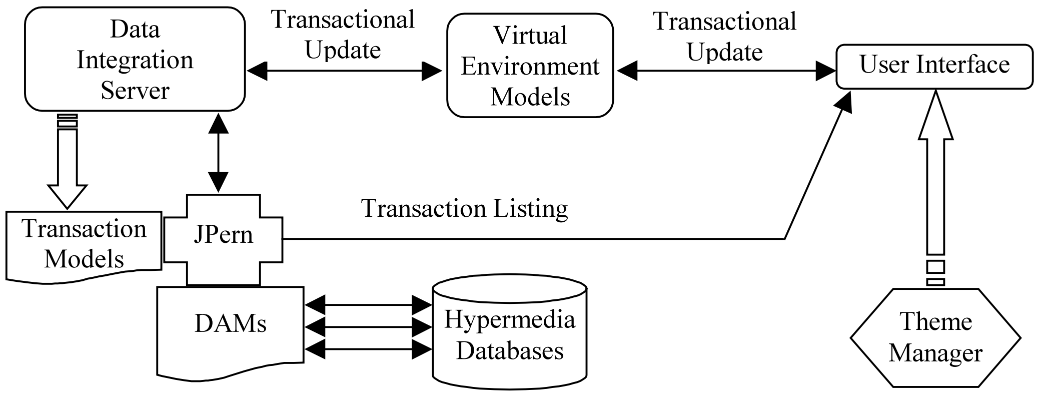

Figure 5 JPern with DIS

The collaboration among users is written in Java, using the transaction model plug-in API provided by JPern. The plug-in code is written and loaded to JPern by administrator of CHIME. When a data object is updated, the change does not go directly to the data source, but rather cached in JPern which stands between DIS and the actual data sources. When any other user requests to refresh their virtual world, the DIS will go through the cache managed by JPern first, and giving them the right version depending on their relationship with the author (or the set of users who currently hold locks on the data object).

5 Summary

As we have shown in this paper, transaction management technologies are found very important in CVE. Not only in traditional areas such as data management, but also in other seemingly less relevant areas such as groupware support and even user interface. A customizable transaction manager such as JPern that can support a wide range of extended transaction models or application specific concurrency control policies gives the necessary flexibility in a component-based system architecture. Also, being able to access and leverage the CC features of the distributed, heterogeneous data sources have shown to be beneficial to the design and construction of web-based CVE. We believe that the application of advanced database technologies in collaborative environment work will generate more fruitful results in the future.

Reference

1. S. Benford, D. Snowdon, A. Colebourne, J. O'Brien and T. Rodden, Informing the Design of Collaborative Virtual Environments**,** *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP'97)*, Nov. 16-19, 1997, Phoenix, AZ, USA. pp. 71-80
2. R. Bentley, W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, S. Sikkel, J. Trevor and G. Woetzel, Basic Support for Cooperative Work on the World Wide Web*, International Journal of Human-Computer Studies 46(6): Special issue on Innovative Applications of the World Wide Web*, p. 827-846, June 1997, Academic Press.
3. P. A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems, Addison-Wesley*, 1987.
4. S. Card, G. Robertson and D. Mackinlay, The Information Visualizer, an Information Workspace. In *Human Factors in Computing Systems Conference,* 1991 pp. 181.
5. P. Ciancarini, A. Knoche, R. Tolksdorf and F. Vitali, PageSpace: An Architecture to Coordinate Distributed Applications on the Web, In *Proc. of the 5th International World Wide Web Conference*, 1996.
6. S. Dossick and G. Kaiser. CHIME: A Metadata-Based Distributed Software Development Environment. To appear in *Joint Seventh European Software Engineering Conference and Seventh ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, September 1999.
7. A. K. Elmagarmid, (Ed.), *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.
8. R. Hubbold, D. Xiao, S. Gibson: MAVERIK - the Manchester Virtual Environment Interface Kernel, (700 Kbytes), *Proceedings of 3rd Eurographics Workshop on Virtual Environments*, Monte-Carlo, February, 1996, European Association for Computer Graphics, P.O. Box 16, 1288 Aire-la-Ville, Switzerland. *Note: An updated version of this paper appears in the published proceedings: Virtual Environments '96, Martin Goebel and Jacques David (Eds), Springer-Verlag, 1996.*
9. IBM, "TXSeries4.2", http://www.software.ibm.com/ts/txseries, 1998.
10. Instinctive Technology, Inc., "eRoom 4.0", http://www.eroom.com/, 1999.
11. M. C. Little, S. K. Shrivastava, S. J. Caughey and D. B. Ingham, Constructing Reliable Web Applications Using Atomic Actions, In *Proc. of the 6th International World Wide Web Conference*, 1997.
12. Microsoft Corp., "Microsoft Transaction Server", http://www.microsoft.com/com/mts.asp, Aug. 1998.
13. Sun Microsystems, "Enterprise JavaBeans™", http://java.sun.com/products/ejb/, July 1998.
14. TIP Working Group, J. Lyon and K. Evans, et al. Transaction Internet Protocol (TIP), 1997, ftp://ftp.ietf.org/internet-drafts/draft-lyon-itp-nodes-04.txt.
15. A. van der Hoek, R. S. Hall, D. Heimbigner, and A. L. Wolf, Software Release Management, Proceedings of the *6th European Software Engineering Conference* , Zurich, Switzerland, Sept. 1997.
16. J. Yang and G. Kaiser, JPernLite: Extensible Transaction Service for World Wide Web, *IEEE Transaction of Knowledge and Data Engineering*, 1999 (In press)