Integrating Transaction Services into Web-based Software Development Environments

Jack Jingshuang Yang, Gail Kaiser, Steve Dossick and Wenyu Jiang
Computer Science Department
Columbia University
USA
{jyang, kaiser, sdossick, wenyu}@cs.columbia.edu

Abstract

Software Development Environments (SDE) require sophisticated database transaction models due to the long-duration, interactive, and cooperative nature of the software engineering activities. Such Extended Transaction Models (ETM) have been proposed and implemented by building application-specific databases for the SDEs.

With the development of World Wide Web (WWW), there have been a number of efforts to build SDEs on top of the WWW. Using web servers as the databases to store the software artifacts provided us with a new challenge: how to implement the ETMs in such web-based SDEs without requiring the web servers to be customized specifically according to the application domains of the SDEs.

This paper presents our experiences of integrating transaction services into web based SDEs. We evolved from the traditional approach of building a transaction management component that operated on top of a dedicated database to the external transaction server approach. A transaction server, called JPernLite, was built to operate independently of the web servers and provide the necessary extensibility for SDEs to implement their ETMs. The transaction server can be integrated into the SDE via a number of interfaces, and we discuss the pros and cons of each alternative in detail.

Topics: Middleware, Groupware/CSCW

Keywords: Transaction Service, Extended Transaction Model, Software Development Environment, Component Integration

1 Introduction

Software Development Environments (SDEs) provide a wide range of services to their users, such as software processe enactment, automation, scheduling, communication between collaborative users, and distributed tool services and data repository services. Accesses to shared data must be moderated by the SDE's concurrency control (CC) component, usually called the transaction manager. Traditionally, the transaction manager is part of the database used by the SDE.

Traditional databases normally enact the serializable, atomic transaction model [Bernstein87]: all the operations in one transaction must be executed successfully or nothing is done. This transaction model was developed to fulfill the needs from application domains such as banking, where updates are frequent and fast, and transactions are short. However it has been found insufficient for application domains such as software engineering and CAD/CAM. In the past decade, researchers have proposed a number of Extended Transaction Models (ETM) [Elmagarmid92] that extend the atomic transaction model in a variety of ways. Such extensions include ensuring application specific data consistency rules rather than traditional serializability, providing finer control of transaction operations rather than atomicity, and allowing collaborative users to share data rather than isolating their transactions.

In the past, such ETMs have been implemented by customizing the underling databases used by the SDEs, by modifying the transaction managers in the databases. This approach was adopted by the SDE developers in the early days [Barghouti92], partly because they used object-oriented model and needed object-oriented databases (OODB), but there was no commercial OODBs available or the available OODBs offered much poorer functionality than the they wanted.

With the development of OODB technology, more and more SDE developers found that they wanted to take advantage of the commercial OODBs for their robustness and to avoid duplicated effort in developing their own OODBs. But the transaction model supported by those commercial OODBs is still the traditional atomic one. In an previous work of our lab [Heineman95], we developed the technology to implement ETMs by adding an extra transaction management layer on top of the underling database system. The transaction management layer utilizes the atomic transaction facilities provided by the underling database, provides a flexible programming interface for the SDEs to customize the CC policies to implement the desired ETM. Such SDEs have a dedicated database that tightly couples with the transaction management layer.

The World Wide Web (WWW) adds more opportunities and challenges to SDE development. A huge number of web servers (over 2.2million servers according to [Netcraft98]) and web pages emerge on the ubiquitous and distributed infrastructure, accessible via a simple and efficient communication protocol HTTP. Therefore, the WWW has been an attractive underling infrastructure for SDE developers [Baentsch95, Dossick95, Bentley97, Miller98]. However, although a few web servers support write (PUT or POST) operations, most of them do not have the basic CC capabilities such as locking and versioning, let alone support for distributed transaction operations such as two phase commit which are important to SDE developers. Because different SDEs need different CC policies to coordinate their users and have different "views" [Yang95] of the web objects, web servers should not hard-wire any specific CC policies so that they can provide a better accessibility to various SDEs.

This paper presents our experiences in integrating transaction services into web-based SDEs. In an earlier work of our lab, we developed a web-based SDE, called WebCity [Jiang97], with a built-in transaction manager component, and a dedicated database. Artifacts stored on the web servers are cached in the local database, and concurrency control policies are enforced for operations of those locally stored web objects. Recently, we developed an external transaction server technology [Yang98], where an independent transaction server, called JPernLite, is responsible for enacting the CC policies, and the data accessed by the SDE are stored in web servers on the WWW.

We integrated the transaction server into our GroupSpace framework for collaborative environments [Kaiser98] and experimented with two integration models: a loose integration model where the transaction services are provided by the independent JPernLite server which enacts the CC policies, and a tight integration model where JPernLite serves as the transaction management component of the SDE.

In Section 2, we describe how we integrated transaction services in WebCity. From this background information, we show some problems of such architecture and in Section 3 we describe our transaction server solution. In Section 4 we present our experiences and evaluation of the integration alternatives. Finally, Section 5 concludes with performance statistics and analysis.

2 Built-in Transaction Manager in WebCity

WebCity is an instance of our web-based, process centered SDE framework Oz [Ben-Shaul93]. Oz is a generic SDE framework that can be instantiated by defining the specific software process of an SDE via a rule-based formalism. WebCity supports the design, coding and testing phases of software development, with rules such as check-in and checkout the source code, edit, compile, link, code review, debug, review the design documents, update design documents, finding the related documentation, manuals and so forth. For simplicity, we will use WebCity to refer to both the framework and the instantiated SDE in the rest of this paper.

In WebCity, software artifacts such as design documents and C source files can be originated from the WWW. WebCity models and caches the web originated documents in a local, dedicated OODB called Darkover. We added a special class called WebObject in Darkover. Whenever an object of class WebObject or any of its subclasses is read, Darkover checks its local cache and fetches the content of the web object if necessary. Similarly, when a web object is written, Darkover automatically uploads it to the originating web server via HTTP PUT.

The transaction management layer of WebCity, called Pern, exploits the locking and recovery features of Darkover. For example, Darkover offers an "undo" interface: each data operation (create, delete, move, set_attribute etc.) in Darkover is given a unique ID. Whenever possible, Darkover can undo an action given its ID.

To implement sophisticated ETMs, Pern provides a "callback" customization interface. Each transaction operation, such as begin, abort, commit of a transaction or lock, unlock of an object, has a "before" and an "after" callback. For example, the before callback of the "lock" operation is called the "lock_before" callback. If any of such callbacks are defined, they will be executed before or after a transaction operation. The callbacks can alter the execution of the transaction operations by telling the transaction operations to return certain status, changing internal state information of the transactions or even sending notification messages to clients. By default, Pern implements the atomic transaction model, but with the callbacks, one can implement sophisticated ETMs. Examples of implementing Altruistic Locking and Epsilon Serializability are described in [Heineman96].

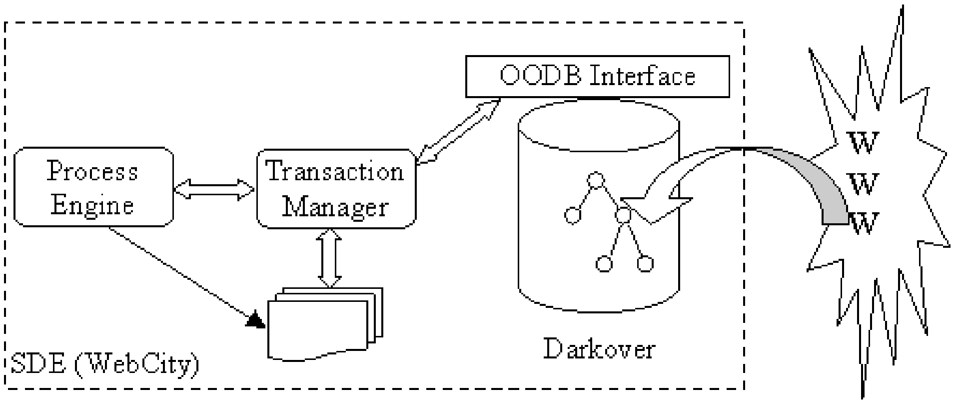

Figure 1. Integrating Transaction Services in WebCity

Figure 1 shows the architecture of WebCity, which consists of a Process Engine, a Transaction Manager and an OODB. The WWW acts as a "dumb" storage facility, serving the documents. The dedicated OODB, Darkover, models and caches the web objects, and provide locking, undo services on those "shadow" objects. The transaction manager Pern utilizes the specific features provided by the underling OODB to offer transaction services, and finally the process engine customizes Pern by using callbacks. This depicts our first attempt to build web-based SDEs. Although WebCity was operational, we found that the architecture was not flexible or powerful enough in general.

First of all, since the database (Darkover) does not know the CC features that a web server provides, the transaction manager could not exploit those features via the OODB. For example, the internet drafts WEBDAV [WEBDAV97] and TIP [TIP97] proposed protocols for the web servers to support locking, versioning and distributed transaction operations. In order to exploit such new features of the web servers, we must extend the architecture so that the transaction manager can be customized to use whatever the CC features available for web objects from different web servers.

Second, by using and modifying Darkover, our own OODB, we gain a lot of advantages by adding specific functionality and interface into the database to handle web objects. This is not generally true if an SDE is based on a commercial database.

Last, but not least, the transaction manager Pern is tightly integrated in the SDE so that it is dedicated to WebCity, being customized to support the transaction model needed by WebCity. Therefore, if multiple SDEs want to share data stored in the same web servers, they have to know each other in advance, and customize their transaction managers to coordinate with each other a priori.

3 Extensible Transaction Server

In order to solve the problems we mentioned in Section 2, we developed a novel transaction server architecture for web-based SDEs. The core of the architecture is a transaction server that operates independently of the web servers and SDEs. The transaction server directly accesses the web servers, therefore it has the opportunity to leverage possible CC features provided by the web servers. The transaction server can also be customized by the SDEs 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 web servers.

3.1 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 [TIP97] 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 [Ciancarini96, Transarc96, Little97] 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 SDEs. Hardwiring CC policies into either the server or the client greatly limits scalability and flexibility.

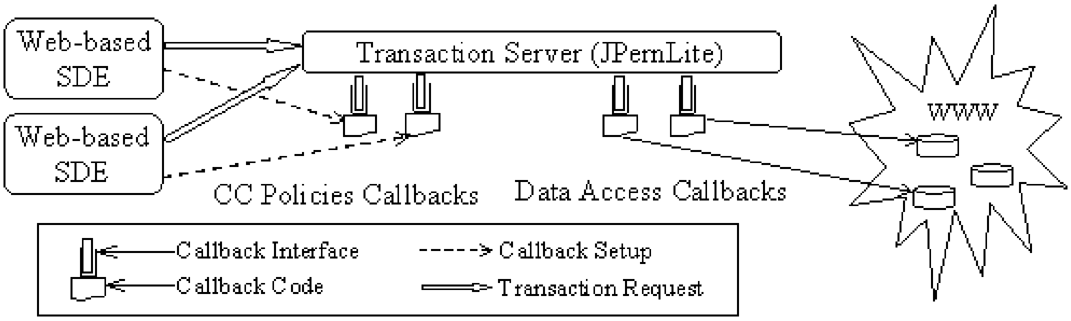

Figure 2. External Transaction Server Architecture

Figure 2 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 SDEs. 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 that the SDEs want.
- The approach does not require any changes to the servers, and the SDEs that use it only need to know how to speak HTTP. On the other hand, the CC features provided by the web servers can be exploited by the transaction server to accomplish more sophisticated ETMs.
- Coordination among SDEs that share data can be done in the transaction server. Furthermore, 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 JPernLite. The name "JPernLite" consists of three parts: "J" for the transaction server is written in Java, "Pern", for we inherited the callback ideas from our previous transaction manager, "Lite" for the lack of a configuration scripting language, which is in our future work. JPernLite can be treated as a normal web server where special URLs are treated as transaction requests. Figure 3 shows sample request and reply messages between a potential SDE and JPernLite.

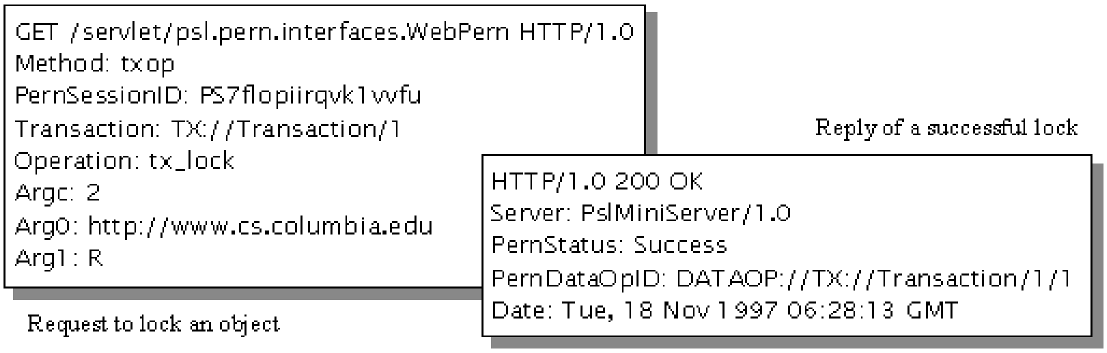

Figure 3. Sample Communication between JPernLite and SDE

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 JPernLite. The reply message shows the successful execution of such a request, and the message tells the SDE that the object has been locked, and the lock ID is `DATAOP://TX://Transaction/1/1`.

3.2 Customizing JPernLite

The CC policies of JPernLite can be customized to implement the ETMs that the SDEs desire. The customization can be done in a number of areas.

- Lock Table: JPernLite uses a lock compatibility table to describe the different lock modes used by the ETM it implements. The default lock table contains the two standard modes: Read-only and read-Write. Additional lock modes can be added, and their mutual compatibility can be customized by setting the lock table entries correctly. JPernLite will automatically carry out the lock compatibility when there are multiple lock requests on an object.
- Lock Semantics: The fine-grained semantics of each lock mode is defined in the callback code that the administrators of each SDEs write. Such callback code can be transferred to JPernLite dynamically provided that the JPernLite server recognizes the SDE. The callbacks are similar to those of Pern [Heineman95], and we give an example of how one might implement two-phase locking in Figure 4. In the "unlock_after" callback, we set the flag "started_unlock" to true, and check the flag before a lock is attempted (in "lock_before" callback).

```
public CBresult tx_lock_before() {
   if (started_unlock) {
      TxOpResult res;
      res=new TxOpResult(TxOpResult.ERROR,
               "PernError=Cannot lock after started unlock");
      return new CBresult (res);
   } else return new CBresult();
}

public CBresult tx_unlock_after() {
   started_unlock = true;
   return new CBresult();
}
```

Figure 4. Callback Code for 2-Phase Locking

- Transaction Modeling: The ETMs are implemented by a combination of customizing the lock table and writing the necessary callback code. For example, to implement an "access-notification" feature that lets a transaction to be notified whenever other transactions access some specific objects. This is done by adding a third lock mode "E" to the lock table shown in Figure 5. In Figure 5, R, W, E are the three lock modes, Read-only, read-Write, and Event-notification, respectively. Y indicates two lock modes are compatible and N the opposite. As the table shows, E mode does not conflict with any other modes, therefore it does not interfer with the normal semantics of the R and W locks. In the "lock_after" callback, one can send a notification to the owner user of the E lock on the object.

	R	W	E
R	Y	N	Y
W	N	N	Y
E	Y	Y	Y

Figure 5. Lock Table for Access Notification

- Exploiting CC Features provided by the Web Servers: JPernLite also has a number of callbacks that the SDE administrator can set up to access the web servers. Such callbacks can be used to allow JPernLite 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.

4 Integration Alternatives

Given the transaction server architecture and the flexibility that JPernLite offers, we integrated the transaction service into our GroupSpace collaborative work framework [Kaiser98] to build an SDE. The SDE is not yet fully functional because the sophisticated rule chaining mechanisms are still under construction. However, we gain a lot of experience in integrating the transaction service into the framework, working with a weaker version of the process engine.

We categorize the integration alternatives according to how tightly they are integrated with the SDE. A loose integration model separates the transaction server from the SDEs. The SDEs make transaction requests to the transaction server through HTTP. The transaction server can either act as a normal web server or a proxy server. A tight integration model that matches the traditional way of integrating transaction services to SDEs, namely the transaction server is degraded to a component of the SDE. In this section, we discuss the pros and cons of each model, and show some of the actual APIs of such integration.

4.1 Loose Integration (1): Independent Server

Running JPernLite as an independent web server is the most natural way of integrating it into SDE frameworks. Figure 2 and Figure 3 showed the architecture and the actual communication between the SDE and JPernLite running as an independent server.

The independent server does not require the SDEs to use HTTP as the communication protocol. In fact, either well known distributed computing techniques such as Java RMI and CORBA, or application specific communication protocols are all plausible candidates. The interface exposed to the SDE via such communication channel contains various transaction operations, such as begin, abort, commit, of a transaction and lock, unlock of an object; customization interface, such as setting up the lock table, CC policy callbacks, and data access callbacks.

4.2 Loose Integration (2): Proxy Server

The independent server approach provides the SDEs a simple interface. However, it is sometimes desired to have the transaction server to act as a web proxy server in order to gain transparency and the capability to chain with other web proxies. A web proxy server accepts requests of web pages from the clients (which are the SDEs in our context), redirects the request to the real web server that serves the page and sends the page back to the clients.

The actual communication among SDEs and the proxy JPernLite server looks very similar to the independent server approach. The SDEs issue an HTTP proxy request instead of a normal HTTP GET request, and the proxy server transparently applies various transaction operations to the data access requests from the SDE.

For example, when the SDE issues a GET request, the transaction server automatically tries to lock the web page in read-only mode; similarly, PUT requests are accompanied by read-write lock attempts. Transaction information can be passed to the proxy transaction server via additional headers similar to the independent server approach. The proxy transaction server can be made to daisy chain through other proxies as well. Figure 6 shows how a proxy transaction server tunnels through a firewall by daisy chains into a firewall proxy server.

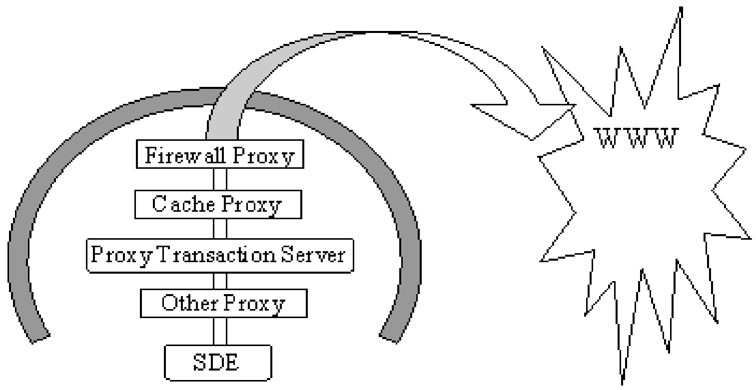

Figure 6. Daisy Chain of Proxy Servers

In short, the additional complexity of implementing the HTTP proxy protocol buys the SDEs the capability to apply transaction operations transparently to the data access requests and the capability to go through other proxy servers for any special purposes.

4.3 Tight Integration: Java Component

The JPernLite transaction server can also be used as a Java component, integrated to the SDE tightly via Java APIs. In our GroupSpace framework for SDEs, the JPernLite fulfills the role of a transaction manager, providing transaction services via a Java interface. The Java API includes methods to request transaction operations, customize the transaction server by setting the lock table, and supply the callback code.

In this approach, customization needs not to be done in the same way that is done in the loose integration models. That is, the SDE can directly extend the `Transaction` class to create new transaction types, which carry the desired callback code. Similarly, accessing different types of web servers that provide different CC capabilities is done by extending the `DataObject` class, so that the `ExternalOp` method of the extended class knows how to contact the specific kinds of web servers to exploit their CC capabilities. Lock table is also directly configured to JPernLite by giving it the file path to the lock table file.

Such tightly integrated transaction server loses the opportunity to coordinate multiple SDEs sharing the web pages stored in the same web servers. Similar to our lab's previous work in federating software process engines [Ben-Shaul95], we plan to extend the transaction servers so that they can establish alliances among each other. Of course, the performance of such tightly integrated transaction servers is much better: the network traffic between the SDEs and the transaction server now becomes local Java calls.

However, we are not sure if such performance improvement have any observable effects in SDEs. The normal tasks of a SDE user lasts for hours and days, and if each task, such as an editing or compiling, issues a couple transaction operations, the time spent on the network traffic can be neglegible.

Another advantage of the tightly integrated approach is security. In the loosely integrated model, the transaction server "trusts" all the recognized SDE administrators so that they can send callback codes for the transaction server to execute. If the SDE tightly integrates the transaction server, it does not need to risk the possible bugs from the other SDEs' callback code that might affect their own data integrity.

5 Performance and Conclusion

5.1 Performance Simulation

In order to analyze the performance of our JPernLite transaction, we performed a series of simulation tests. The response of each transaction operation consists of two parts: network traffic and processing time. Because the roundtrip time on the network depends on the condition of the internet (or LAN if the SDE and the transaction server run in the same local network), the simulating the network traffic part would not be of our interest. One interested in such network traffic statistics should turn to internet surveys that concerning the performance of regular HTTP servers.

Therefore, we focused on the processing time of each transaction request. In order to simulate the transactions issued by N users, we randomly generated N sessions, each session randomly generated a sequence of operations, such as starting transaction, locking data objects, reading/writing objects and so on. The length of each session is randomly decided by whether or not a session starts a new transaction after the current one aborts or commits. To simulate the real-world situation where a small set of data is accessed much more frequently than others, we also selected the a subset of the data to be accessed much more frequently by the transactions.

We simulated the cases of 10, 100, 1000 users using a single JPernLite server. The average process time for a single transaction operation are 120, 145, 450 milliseconds, respectively. The increase of average processing time results from the increasingly large data structures used by JPernLite to maintain the status of the transactions. With larger data structures, the time used to access and update each transaction becomes larger. Also, the interactions among the transactions due to the share of data objects become more complicated with larger number of transactions. The statistics turns out to be quite satisfactory: the process time grows much slower than the number of users, thus showing JPernLite's potential scalability.

5.2 Conclusion

We have presented our experiences in integrating transaction services into web-based SDEs through our WebCity and JPernLite development experiences. WebCity adopts a traditional approach where transaction manager is built as part of the SDE, utilizing a local database which stores shadow copies of web objects. The CC policies of the SDE is applied to the shadow objects in the local database. Sharing and collaboration among SDEs are difficult and hard to accomplish.

JPernLite, our extensible transaction server, provides transaction services to SDEs independently of the web servers. It uses the callback mechanism to customize its CC policies and data access modules which exploits the possible CC features the web servers provide.

We also investigated three possible ways to integrate JPernLite into SDE frameworks. The loose integration models integrate JPernLite as either a web server or a proxy server, providing the opportunity to coordinate multiple SDEs. The tight integration model helps improving performances and has better security control. We also simulated the situation of a large number of users using the transaction server, and the statistics showed that the JPernLite server we implemented scales up quite well.

Reference

[Bernstein87] P. A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems, Addison-Wesley*, 1987.

[Elmagarmid92] A. K. Elmagarmid, (Ed.), *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.

[Barghouti92] N. S. Barghouti, Supporting Cooperation in the Marvel Process-Centered SDE, In *Proc. of 5th ACM SIGSOFT Symposium on Software Development Environments,* pp. 21-31, December 1992

[Ben-Shaul93] I. Z. Ben-Shaul and G. E. Kaiser and G. T. Heineman, An Architecture for Multi-User Software Development Environments, *Computing Systems The Journal of the USENIX Association*, 6 (2), 65-103, Spring 1993.

[Heineman95] G. Heineman and G. Kaiser, An Architecture for Integrating Concurrency Control into Environment Frameworks, In *Proc. of the 17th International Conference on Software Engineering*, 1995.

[Ben-Shaul95] I. Ben-Shaul and G. E. Kaiser, *A Paradigm for Decentralized Process Modeling*, Kluwer Academic Publishers, Boston MA, 1995.

[Baentsch95] M. Baentsch, G. Molter and P. Sturm, WebMake: Integrating Distributed Software Development in a Structure-Enhanced Web, *In Proc. of the 3rd International Conference of WWW*. Darmstadt, Germany, April 1995,
http://www.igd.fhg.de/www/www95/papers/51/WebMake/WebMake.html

[Dossick95] S. E. Dossick and G. E. Kaiser, WWW Access to Legacy Client/Server Applications, In *Proc. of the 5th. International WWW Conference,* pp. 931-940, Paris, France, May 1995.

[Yang95] J. J. Yang and G. E. Kaiser, An Architecture for Integrating OODBs with WWW, In *Proc. of the 5th. International WWW Conference*, pp. 1243-1254, Paris, France, May 1996

[Transarc96] Transarc Corp., Transarc DE-Light Web Client Technical Description, February 1996, http://www.transarc.com/afs/transarc.com/public/www/Public/ProdServ/WWW/delov.html.

[Ciancarini96] 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.

[Heineman96] G. T. Heineman, A Transaction Manager Component for Cooperative Transaction Models, Ph.D. Thesis, CUCS-010-96, Columbia University Department of Computer Science, June 1996.

[Little97] 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.

[Bentley97] 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*, Academic Press, 1997.

[Jiang97] W. Jiang, G. E. Kaiser, J. J. Yang and S. E. Dossick, WebCity: A WWW-based Hypermedia Environment for Software Development, In *Proc. of 7th Workshop on Information Technologies and Systems*, pp 241-245 December 1997.

[TIP97] 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.

[WEBDAV97] WEBDAV Working Group, Y. Goland, E. J. Whitehead, A. Faizi, S. Carter, et al. Extensions for Distributed Authoring and Versioning on the WWW WEBDAV, 1997, ftp://ds.internic.net/internet-drafts/draft-jensen-WEBDAV-ext-01.txt.

[Netcraft98] Netcraft Corp., The Netcraft Web Server Survey, 1998, http://www.netcraft.com/survey/.

[Miller98] J. Miller, D. Palaniswami, A. Sheth, K. Kochut and H. Singh, WebWork: METEOR's Web-based Workflow Management System, *Journal of Intelligent Information Systems*, 10, 2, March 1998. (in press)

[Yang98] J. J. Yang and G. E. Kaiser, JPernLite: An Extensible Transaction Server for the World Wide Web, In *Proc. of the ACM Conference on Hypertext and Hypermedia*, 1998. (To appear)

[Kaiser98] G. E. Kaiser, S. E. Dossick, W. Jiang, J. J. Yang and S. X. Ye, WWW-based Collaboration Environments with Distributed Tool Services, *Journal of World Wide Web*, 1, pp.3-25,1998, Baltzer Science Publishers.