A Tree-Structured Persistence Server for Data Management of Collaborative Applications

Chien-Min Wang, Hsi-Min Chen, Gen-Cher Lee, Shun-Te Wang, and Shyn-Fong Hong
Institute of Information Science, Academia Sinica
NanKang, Taipei, Taiwan
Email: {cmwang, seeme, gc, wangsd, fong}@iis.sinica.edu.tw

Abstract

The persistence problem of collaborative applications is an essential issue in the research of computer-supported collaborative work. A collaborative computing environment requires a simple and transparent persistence middleware to deal with complex data accesses. Therefore, in this paper, we propose a data persistence mechanism and implement a persistence server, called Tree-Structured Persistence Server (TSPS), to support the data management for collaborative applications. The TSPS allows states of collaborative applications to be stored in a tree fashion beside tables. By introducing TSPS, the problems of collaborative applications, such as persistence data access, asynchronous collaboration, latecomer, version control, etc., can be solved easily. The TSPS was originally developed to serve as a persistence layer to support a project of computer-supported collaborative work. We intend to develop our persistence mechanism for universal use so that it can be applied in more application areas.

1. Introduction

As the need for applications capable of working collaboratively has increased, more and more researchers have devoted their attentions to the development of Computer-Supported Cooperative Work (CSCW). However, persistence issues of collaborative applications for CSCW have not been given enough attention [1]. The problems of collaborative applications, such as persistence data access, asynchronous collaboration, latecomer, version control, etc., can be solved easily, if the persistence techniques can support CSCW properly. Consequently, we propose a persistence mechanism to deal with the data persistence problem in CSCW.

Depending on temporal dimensions, CSCW applications are classified into two broad categories, synchronous and asynchronous collaborations. Where users are separated geographically, they can find a common time to meet in through synchronous collaboration. On the other hand, users might not be able to work simultaneously because of different time zones, or different domain knowledge. Asynchronous collaboration allows these users to cooperate when temporally separated. In order to integrate synchronous and asynchronous collaborations, collaborative persistence techniques are necessary, as they can bridge the gap between these two kinds of collaborations.

For CSCW applications, the primitive functionality of data persistence is to save and restore their run-time states. On the other hand, we usually utilize version control tools to keep track of the changed records of applications that we develop. The developers of CSCW applications might like to archive application states in a tree-structured manner to keep changes rather than overriding them.

The above observations led us to develop a data persistence implement. We derived a persistence mechanism and implemented a Tree-Structured Persistence Server (TSPS) helping users to access persistent data transparently and simply. We expect that it can be applied in other application areas in addition to CSCW, so we develop it for universal use. This paper primarily describes our proposed mechanism and its implementation for managing persistence data of collaborative applications.

2. Background

2.1 ShareTone Project

TSPS was originally developed as a persistence middleware to support a CSCW project, called ShareTone [2]. The ShareTone project is a Java-based collaborative computing platform in which dispersed users can work together. The term "collaborative" means that when one member of a cooperative group changes some states of operated objects in his/her collaborative application, the other members will see the same changed effect in their applications simultaneously, i.e. What You See Is What I see (WYSIWIS) [3]. By using ShareTone, the component developers can tailor standard JavaBeans components, or write their own beans with some extra codes, to make these beans capable of working collaboratively. The application developers can use these collaborative beans

Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)
1550-445X/05 $20.00 © 2005 **IEEE**

to assemble collaborative applications, such as collaborative whiteboards, collaborative editors, chat rooms, etc., on the top of the ShareTone platform. Users can, therefore, use these collaborative applications to work cooperatively.

2.2 Features of Persistence Data

Before developing TSPS, we elicit the data persistence requirements from ShareTone project initially. The fundamental need is a persistence store that allows collaborative applications to store and retrieve their states. Besides, there are several features, e.g. *session, checkpoint, undo and redo, latecomer, versioning,* and *query*, those TSPS needs to support for collaborative applications. Figure 1 shows the layered model of the collaborative computing environment.

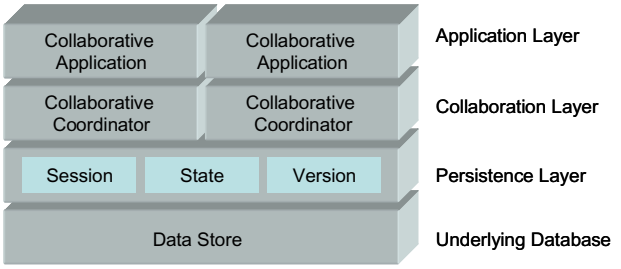

Figure 1. A collaborative computing environment mode

2.3 Related Work

There are a number of persistence mechanisms, including implementations and specifications, are proposed to deal with data or object persistence in various fields. *JDBC*, *EJB* [4] and *JDO*[5] are data/object persistence techniques used in various levels of persistence data for different requirements. Since these have their own advantages in different applications and domains, developers can choose a suitable one according to the persistence context. However, the limitations of these techniques, such as encapsulated granularity, code intrusions, and overheads, make them unable to meet the persistence requirements of collaborative applications properly. The lack of persistence supports of these techniques leads us to develop a persistence mechanism and its implementation, which realizes the features mentioned above.

3. Design Choices

We consider design choices in advance of developing the TSPS. There are two significant issues to be addressed. First, we must appraise what kind of object serialization techniques is proper for the persistence server. Second, since application states must be stored in a database, we have to choose an appropriate database for the persistence server.

Java Object Serialization is the process of marshaling a run-time object state into a sequence of bytes or a form of text, as well as the process of recovering serialized bytes or a text into a live object at some future time. There are three main serialization mechanisms: *Java Serialization API*, *Java Long-Term Persistence* [6], and some native persistence techniques [7], which are intended to deal with object serialization problems. After considering these three mechanisms, we decide to use *Java Serialization API*. Although its data size is larger, the information of a serialized object captured in this manner is more complete than the others.

We also studied the features of back-end databases in detail to assess its suitability. The chosen database has to satisfy the needs of sessions, versions, and query from the previous features. Candidates include three popular types of database: *Relational Database*, *Object-Oriented Database* [8] and *Native XML Database* [9]. We decided to use *Native XML Database* (NXD) as our back-end database. NXD is a lightweight database that consumes fewer resources than other systems.

4. Architecture of the Tree-Structured Persistence Server

The TSPS was originally developed to manage persistence data of computer-supported collaborative work. The tree-structured means that the persistent data of collaborative applications can be stored in a tree fashion. The architecture of TSPS consists of eight components as shown in Figure 2. In the following, we will take a detailed look at these components.

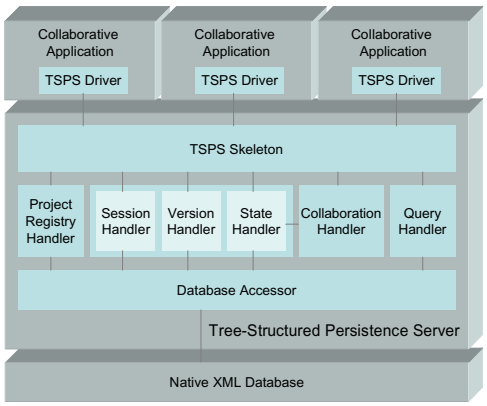

Figure 2. Architecture of the TSPS

(1) **TSPS Driver**: is a client-side proxy that connects to the server over the Internet. To provide client-side developers with consistent interfaces, we intend to implement the driver simply and flexibly. As a result, we developed the TSPS driver by referring to design

patterns [10]. Consequently, developers can request services according to the demand of communication technologies.

(2) **Project Registry Handler**: stores project information of collaborative applications. After registration, other participants can look collaboration projects up from the handler and choose a desired one to join. Through the project registry handler, collaboration project information can be archived permanently. The project registry handler keeps a list of projects including created, working, and closed ones.

(3) **State Handler**: facilitates the access of application states. When a user joins a session, he/she can save persistent objects into the server while filling out associated information in the header for identification. With header information, users can understand the contexts of persistent objects and retrieve them from the server. Storing application states sequentially in each persistence session forms a state tree as shown in Figure 3. A developer who wants to build an application can set checkpoints in every development step by storing states. Sometimes the developer, who might like to undo the effect of the current operations, can recover a preceding state of a corresponding checkpoint through retrieval. The developer can also take redo actions based on state headers to retrieve corresponding states.

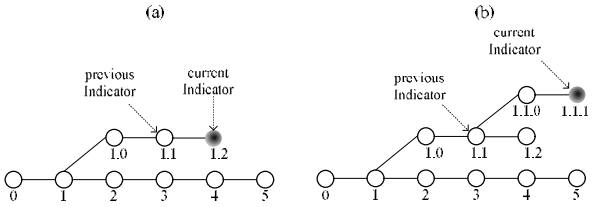

Figure 3. (a) Storage of a state if the previous state is the last one; (b) Storage of a state if the previous state is not the last one.

(4) **Version Handler**: is responsible for managing branches of a state tree. When users store a series of states during the development and the execution of applications, a state tree is shaped. The version handler can arrange states in a tree structure to control versions. Figure 3 shows an example of a state tree. Users can traverse a state tree in each session by using a branch browser. Each session has an *Indicator* to denote the current work state and show the corresponding branch. Through the branch browser, users can set an *Indicator* that specifies the current work state and branch.

(5) **Collaboration Handler**: is responsible for the concurrency control when multiple users access the same persistence data simultaneously. It is essential for collaborative applications to provide concurrency control mechanisms that help users avoid conflicts when manipulating the same object. As collaborative applications, TSPS suffers the same problem. The collaboration handler provides an object-based floor control to overcome the problem of concurrency access. It allows users to bind resources with a floor by themselves. Before accessing these resources, users have to request and gain the access right of the associate floor from the collaboration handler. Users without the access right cannot perform the saving or retrieving action on these resources. Through this way, we can prevent access conflicts in TSPS. Figure 4 shows the collaboration behavior in TSPS. When member *A* got a floor saves the current state into TSPS, the save action is executed in the local but the actions, such as updating state tree structure, setting current indicator, setting current branch, etc., are designed to be broadcasted as shown in Figure 4(a). Retrieval action is similar to saving. The difference is that TSPS has to restore retrieved state to each collaborative application as shown in Figure 4(b).

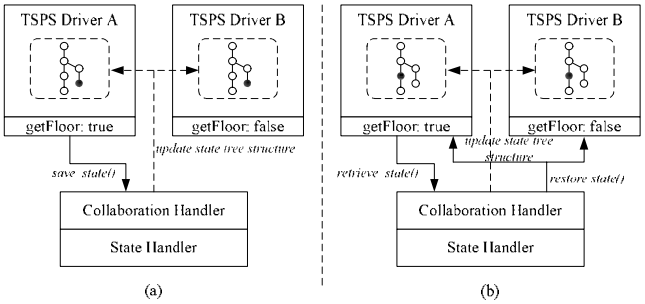

Figure 4. The collaboration behavior of TSPS

(6) **Query Handler**: deals with queries related to sessions, states, and branches in the TSPS. Through it, users can find appropriate sessions, states, and branches.

(7) **Database Accessor**: is responsible for constructing connections between the TSPS and the back-end database, i.e. NXD, and providing other components with interfaces to access the database. The database accessor wraps up the database so that components, like the session handler and the state handler, can access persistent data transparently. Because of the database accessor, we can utilize various NXD.

5. Application Examples

In this section, we illustrate applications of archiving Java run-time states by employing the TSPS. The application is the ShareTone Composer as shown in Figure 5. It is built on the ShareTone platform and an integrated development environment for developing collaborative applications in Java. The ShareTone Composer carries a TSPS driver so that it can access services. Developers can save the development states as checkpoints. The saved states

logically form a flow from which developers can traverse to undo and redo manipulated actions. An application can be recovered from a previous state by using the state query.

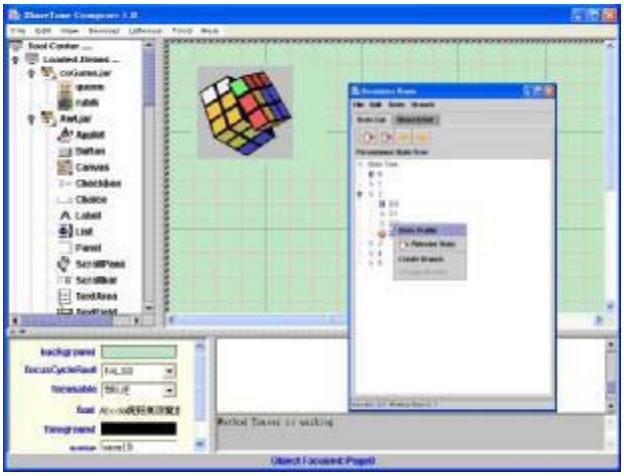

Figure 5. ShareTone Composer

The second application is ShareTone CoEditor, as shown in Figure 6, which allows separate users to edit different paragraphs simultaneously in the same article. Each user has his/her own cursor, called a telepointer, to tell others where they are in the article. This helps to avoid conflict. Each user also has his/her own color to distinguish the words they marked. As with the ShareTone Composer, the ShareTone CoEditor integrates our TSPS driver and thereby access persistence services.

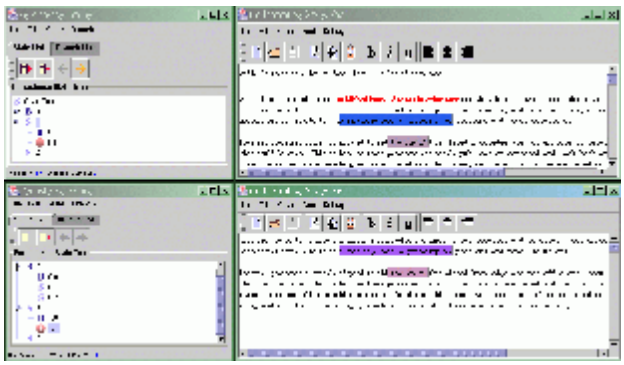

Figure 6. ShareTone CoEditor

6. Conclusions and Future Work

A collaborative computing environment requires a simple and transparent persistence mechanism to deal with complex data access. In this paper, we proposed a tree-structured persistence mechanism and implemented it to support collaborative applications to manage persistence objects. The implemented TSPS is a light-weight management system for persistence objects. It serves as a persistence layer by which the front-end application can manage its persistence objects. The TSPS provides three kinds of services, i.e. session service, state service, and version service. The session service can manage created sessions, the state service can store and retrieve persistent objects into/from the store and the version service can create various versions of applications by committing application states in branches. The TSPS is not only suitable for CSCW applications, but it is developed for universal use so that it can be applied in several application areas. In the near future, our research efforts will focus on caching and transaction mechanisms.

Acknowledgements

This work was supported in part by National Science Council under Contract Nos. NSC92-2213-E-001-015 and NSC93-2213-E-001-008.

References

[1] J. A. Mariani, and T. Rodden, "The Impact of CSCW on Database Technology," *in proceedings of IFIP International Workshop on CSCW*, Berlin, Germany, (1991) 146-161.

[2] ShareTone, available at http://www.sharetone.org

[3] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar, "WYSIWIS Revised: Early Experiences with Multiuser Interfaces," *ACM Transactions on Office Information Systems*, Vol. 5. No. 2, (1987) 147-167.

[4] Richard Monson-Haefel, *Enterprise JavaBeans*, O'Reilly, Oct. 2001.

[5] David Jordan, and Craig Russell, *Java Data Objects*, O'Reilly, April 2003.

[6] Philip Milne and Kathy Walrath, "Long-Term Persistence for JavaBeans," Available at http://java.sun.com/products/jfc/tsc/articles/persistence

[7] JOX: Java Objects in XML, available at http://www.wutka.com/jox.html

[8] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, "The Object-Oriented Database System Manifesto," *in proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan (1989) 223-40.

[9] Wolfgang Meier, "eXist: An Open Source Native XML Database," *Web, Web-Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops*, Erfurt, Germany, October 2002

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Massachusetts (1994)