A Tree-Structured Persistence Server for Archiving Java Run-Time States

Chien-Min Wang, Hsi-Min Chen, Shun-Te Wang, and Shyn-Fong Hong

Institute of Information Science, Academia Sinica

NanKang, Taipei, Taiwan

Email: {cmwang, seeme, sdwang, fong}@iis.sinica.edu.tw

Abstract

The persistence problem of collaborative applications is a significant issue in the research of computer-supported collaborative work. A collaborative computing environment requires a simple and transparent persistence layer to deal with complex object accesses. Therefore, in this paper, we propose an object persistence mechanism and implement a persistence server, called Tree-Structured Persistence Server (TSPS), to support collaborative applications that store and retrieve application states. The server allows states of collaborative applications to be stored in a tree fashion beside tables. In addition to accommodating basic operations to access persistent objects, the TSPS offers advanced functions to manage sessions, states and versions of a collaborative application. By accessing these services, users can archive the run-time states of Java-based applications with transparency and simplicity. 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.

Keywords

Object persistence, CSCW, Java, object serialization, database, XML

1. Introduction

As the popularity of Internet has grown, users have become accustomed to utilize Internet-based applications to communicate with each other. Since the need for applications capable of working collaboratively has increased, more and more researchers have devoted their attention to the development of Computer-Supported Cooperative Work (CSCW) [1, 2, 3]. It is a technology designed to facilitate team work. The technology may be used to communicate, cooperate, coordinate, solve problems, compete, or negotiate over the network. A number of problems with CSCW have been investigated in the last decade. These can be categorized as follows: (1)

Transparency issues of CSCW: How to spend less effort making stand-alone applications collaborative, as well as keeping the flexibility that allows users configuring shared and private working areas simultaneously [4, 5]. (2) Issues of centralized and replicated shared information: discuss whether the shared object is a single copy operated by multiple users, or each user keeps a replicated one in the local computer [6, 7, 8]. (3) Issues of concurrent control: How to deal with users' inputs that are interleaved and avoid conflicting [5, 8]. (4) Access control issues: explain that users having access rights can manipulate particular shared objects, but users without access rights cannot [9, 10]. (5) Awareness information issues: allow users to be aware of what others are doing [11, 12]. Persistence issues of collaborative applications for CSCW have not been given enough attention [13, 14]. The complexity of the above issues will be reduced, if the persistence techniques can support CSCW properly. We propose a persistence mechanism to solve the object persistence problem in CSCW.

Depending on temporal dimensions, CSCW applications are classified into two broad categories, synchronous and asynchronous collaborations [1]. 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 work simultaneously because of different time zones, or different domain knowledge. For example, one person might reside in U.S., while the other resides in Taiwan, so that it is difficult to communicate at the same time. Or a staff member might have to wait for a business decision made by a senior manager. Asynchronous collaboration allows these users to cooperate when temporally separated. In order to integrate synchronous and asynchronous collaborations, collaborative persistence techniques have become more important, as they can bridge the gap between the tow collaborations.

For CSCW applications, the primitive functionality of object persistence is to save and restore their run-time states. Whether it is a whiteboard application that saves drawings, an editor application that archives text contents or a game that saves the progress of a player, the ability to store states and later retrieve them is a vital function. On the other hand, we usually utilize version control tools to keep track of the development records of applications when we construct them. Therefore the developers of CSCW applications might like to archive application states in a tree structure manner to keep versions rather than overriding them.

The above observations led us to develop an object persistence implement. We derived a persistence mechanism and implemented a Tree-Structured Persistence Server (TSPS), which supports our mechanism and is able to facilitate access persistent objects 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 introduces our proposed mechanism and its implementation for archiving Java run-time states. The rest of the paper is organized as follows. Section 2 lists the development requirements presented to support the object persistence of computer-supported collaborative work. Section 3 discusses our design choices of object serialization techniques and underlying databases. Section 4 explains the architecture and internal components in detail. In session 5, we illustrate applications of archiving Java run-time states by using the TSPS. Session 6 describes related work about other recent persistence mechanisms. Finally, we present some concluding remarks in the last section.

2. Development Requirements

TSPS was originally developed as a persistence layer to support a CSCW project, called ShareTone [15]. The ShareTone project is a Java-based collaborative computing platform in which people at different locations can work together. 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 to compose collaborative applications on the top of the ShareTone platform. For instance, application developers can build a collaborative whiteboard, a collaborative editor, collaborative games, etc. Users can, therefore, use these collaborative applications to 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 changes in their applications simultaneously, i.e. What You See Is What I see (WYSIWIS) [16].

Before developing TSPS, the main task is to elicit the requirements from ShareTone project. The primary need is a persistence store with supporting functions that allows collaborative applications to store and retrieve their states into/from it. The following scenarios should be noted.

- A latecomer might like to join a working session. He/she can select an existing session from a working session list, and catch up with the progress of the session work. This user can then work with other existing users collaboratively.
- Users might like to continue a closed session. They can select a session from a closed session list. The closed session then becomes available for the users who can continue the collaborative work.
- Users might like to set checkpoints so that they can jump to each checkpoint

wherever they set. Users can set their own checkpoint mechanism or they can set checkpoints at any positions on the fly.

- After users manipulate a series of operations, they may like to jump back to a specified checkpoint.
- Users might like create a new version of a collaboration application within a session. For example, users can keep a developing version and a testing version simultaneously. Nowadays there are several products on the markets that provide this kind of mechanism, such as CVS (Concurrent Versions System) [17], Rational Clearcase [18], Microsoft Visual SourceSafe [19] etc. With these, users can create various versions of applications for different purposes.

Based on these above scenarios, we summarized a number of features as follows:

1. **Session**: A session of CSCW can be thought of as a meeting at which participants discuss some issues. In the real world, a chair announces the date and location of a meeting. Then the participants either actively attend or are invited to join. Finally, the meeting is closed after some conclusions. By using "session" function, people can choose to take part in an interesting one and avoid others. The function can also be used to filter out users who don't have proper authorization. Walking through the session lifecycle, users create a session before they begin cooperative work. In the session, the operations can be recorded and at the end of the collaboration, the session can be closed. The closed session can be reopened at a later time to continue the cooperative work. The persistence server must provide functions to deal with session management.
2. **Storing and Retrieving Application States:** In the ShareTone project, collaborative work is divided into two phases, the design phase and the application phase. In the design phase, developers can join a development session and build collaborative applications. Later, developers might like to set a checkpoint to record the current development states, or save current states when they close the session. Subsequently, developers can restore previous states by reopening the session and go back to a certain checkpoint. In the application phase, users join a session to work cooperatively through collaborative applications. In each session, users might like to save the current state of applications they have worked on, as they can be reopened later.

3. **Undo and Redo:** As stand-alone applications, when users make use of collaborative applications to work together, they might like to go back to a previous state and then go forward to the next state. For example, users can write down some words and then undo these actions, or redo the effect by using a word processing application. Therefore, the persistence server must provide functions to undo and redo actions.

4. **Versioning:** Developers might like to build a variety of versions of collaborative applications without creating multiple sessions. A development session should not be restricted one version only. Once developers participate in a development session, they can create various versions of collaborative applications, based on their demands. For example, some versions are used for developing and some are used for testing. Therefore, the persistence server must provide functions to control versions.

5. **Query**: Since a number of sessions exist in the collaborative environment and each session further consists of versions and states, we need to be able to locate an interesting one from a large number of sessions, version and states. The query mechanism is a useful means to help users find an appropriate target according to their preferences. Again, the persistence server must provide functions that support query.

6. **Distributed Computing Environment**: With the growth of the Internet, most applications have developed to become Internet-based ones, especially CSCW applications. Users separated geographically can utilize collaborative applications to work cooperatively via the Internet. Because of the characteristics of distributed computing environments, the persistence server also has to be able to operate on the Internet. For this reason, the persistence server must be implemented as a client/server model so that users manipulating client applications can remotely access the persistence services. Also, the persistence server should accommodate a variety of communication techniques, such as Socket, JMS (Java Message Service), and Web Services, etc., so users can choose a suitable one with ease.

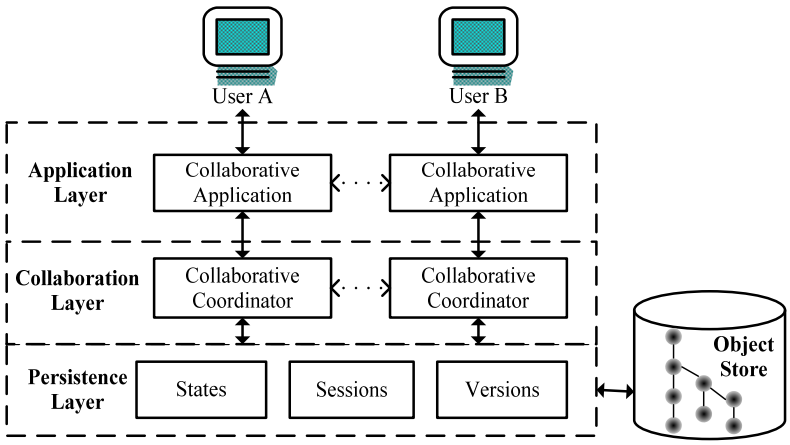

Figure 1. A collaborative computing environment model

Figure 1 shows the model of the collaborative computing environment. Users utilize collaborative applications to work and communicate with each other in the application layer. Collaborative applications that are capable of working together are built on the top of the collaborative environment. The collaboration layer allows applications to share common states to achieve collaboration. The persistence layer connected to the database accommodates applications with persistence support, such as session, state, and version managements.

3. Design Choices

We have to consider design choices before developing the TSPS. There are two main issues we have to address. First, the objects contained in collaborative applications must be serialized and desterilized because the application states have to be stored and recovered across the network. Therefore, we must appraise what kind of object serialization techniques is proper for the persistence server. Second, since the application states have to be stored in a database, we have to chose as appropriate database for the persistence server. Relational database, object-oriented database and other databases have their own advantages and disadvantages respectively. An appropriate database for a persistence server is essential. We will explain these considerations and decisions in the following sections.

3.1 Java Object Serialization

Java Object Serialization [20] 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. Object serialization is usually used in network applications that have to exchange run-time objects. It used to be applied in lightweight persistence and for communication in sockets, Remote Method

Invocation (RMI), JMS and other Java-based communication technologies.

Since most of collaborative computing environments utilize these Java-based communication technologies to communicate among collaborative applications, what object serialization mechanisms that are suitable for our persistence server are very important. A collaborative application comprises a variety of objects. For instance, a simple text editor is composed of Java Swing components, such as JFrame, JPanel, JTextarea, etc. Developers save the instances of these components in a repository for later use. When they want to use the application, again, they can retrieve these instances from the repository and return to the state they saved before. These processes need object serialization mechanisms to transmit run-time objects. There are three main serialization mechanisms: Java Serialization API [21], Java Long-Term Persistence [22], and some native persistence techniques, which are intended to deal with object serialization problems.

- **Java Serialization API:** Provides a standard mechanism for Java developers to handle object serialization. Java developers create an object capable of serialization by implementing the standard Serializable Interface. In addition to single object serialization, Java Serialization API supports the serialization of the whole object tree in which consisted objects are also implemented Serializable one. The result of this serialization mechanism is an array of bytes.

- **Java Long-Term Persistence:** Is used in version 1.4 of J2SE to serialize JavaBeans instances into XML documents, as well as recover these instances from XML documents. Java Long-Term Persistence also supports the serialization of an object tree. Since the XML documents just record the modified properties of JavaBeans instances, instead of dumping the memory into bytes, the size of the serialization document is rather small in contrast to Java Serialization API. The output of this serialization mechanism is an XML document.

- **Native Serialization Techniques:** The above serialization mechanisms are proposed by Sun. However, there are some native persistence techniques presented for different purposes, such as JOX (Java Objects in XML) [23] and KBML (Koala Bean Markup Language) [24].

After considering all the mechanisms, we decide to use Java Serialization API. Although the data size of this one is larger, so far the information of a serialized object captured in this manner is more complete than the others. The main shortcoming of Java Long-Term Persistence is that serialized objects must comply

with the rules of JavaBeans. Thus, some properties and inner classes, such as listener classes, adapter classes, etc., that do not conform to the format of JavaBeans cannot be marshaled into a XML document and cannot be recovered. Nevertheless, serialization based on XML encoding/decoding approaches have become an increasing trend, so we believe that Sun will continue to improve this mechanism to take over form the traditional Java Serialization API in the future. We do not recommend the use of proprietary persistence approaches because of maintenance problems. Java Serialization API is, without doubt, the most suitable mechanism. Note that certain objects such as Thread, Socket, Image, I/O, and its subclasses are not serializable by nature. In order to reconstruct these unserilaizable objects when applications are restored, developers can use a customized approach of Java Serialization API to recover the effects of objects after deserialization.

3.2 Underlying Databases

In order to facilitate the storing and retrieving of application states as well as management functionality, we 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 requirements. The following information describes three popular types of database as candidates.

- **Relational Database:** Relational database is the most used and mature among the candidates. Standard access interfaces are identified and users can manipulate the database by utilizing SQL. Many famous companies have developed a variety of RD-compatible database management systems over a long period of time, e.g. Oracle Database, Microsoft SQL Server and Sybase Database. There are also several open source database management systems available, e.g. MySQL and PostgreSQL.

- **Object-oriented Database:** As the growth of object-oriented programming languages continues, the need for corresponding databases is increasing [25]. Since the translation between object-oriented data and relational concept is complex and time comsuming, object-oriented databases are proposed to process the data of object-oriented programs transparently. A number of OODBMS products, like Gemstone, Objectivity/DB, ObjectStore, etc., have emerged from the market of database systems today [26].

- **Native XML Database:** Native XML Database (NXD) was proposed by XML:DB Initiative [27]. NXD defines a model for an XML document, as opposed to the data in that document, and stores and retrieves documents according to that model [28]. NXD is a pure XML database framework in

which users can store and retrieve XML documents, without the need to convert structures. It also supports XPath [29] and XUpdate to locate and modify the content of a XML document. XML:DB Initiative have defined standard XML database APIs and so a number of implementations have been realized, such as Apache Xindice [30] and eXist [31].

We decided to use NXD as our back-end database. NXD is a lightweight database that consumes fewer resources than other systems. It also stores and retrieves XML documents directly. Applying XML technologies to Internet-based systems will be the pattern for the future. More and more mechanisms of object serialization facilitate object persistence in the XML manner. Also, the tree-structure characteristic of XML is consistent with the tree-structure characteristic of versions from the perspective of version control. Although relational databases are the most popular, programmers need to be familiar with SQL, access APIs and the structures of relational models which are heavy in comparison to the simplicity of NXD. Many object-oriented databases are proprietary with the result that users make use of them incompatibly.

4. Architecture of the Tree-Structured Persistence Server

The TSPS was originally developed as a management system of an object store to support computer-supported collaborative work. The tree-structure means that the persistent objects of collaborative applications can be stored in a tree fashion, in addition to tables. However, we plan to develop TSPS to apply in more application areas, and not restrict it to CSCW only. Therefore, the TSPS is designed to save any objects, or data that can be encoded in a text format. The architecture of TSPS consists of six components as shown in Figure 2, i.e. (1) TSPS driver and skeleton, (2) session handler, (3) state handler, (4) version handler, (5) query handler, and (6) database accessor. In the following subsections, we will describe these components in detail.

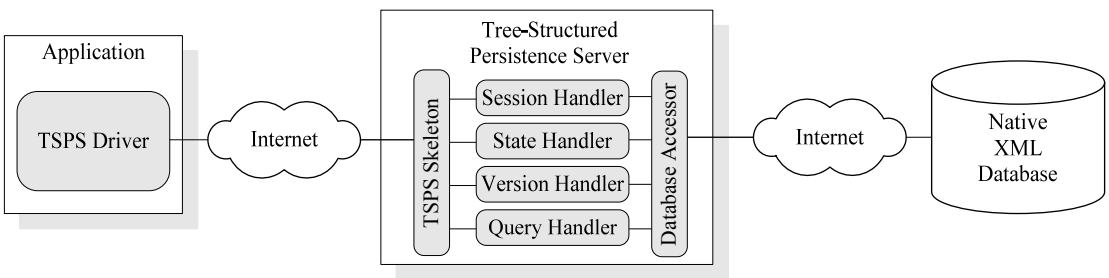

Figure 2. Architecture of TSPS

4.1 TSPS Driver and Skeleton

The 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 [32]. Using Proxy pattern, the developers of collaborative applications can integrate the driver with their programs to access TSPS services, like session, state and version management. Using Façade pattern, applications are able to start a connection, request services and finally release the connection through a single interface furnished by the driver. Using Factory pattern, the TSPS driver can employ present communications technologies, such as Socket, JMS, and Web Services, to communicate with the TSPS. Consequently, developers can request services according to the demand of communication technologies.

4.2 Session Handler

The session handler is responsible for managing sessions created by users. A session consists of a unique identifier generated by the system, a session name, a creator, a creation date and a description. Figure 3 illustrates the lifecycle of a persistence session. Before requesting a persistence server, users must first create a session, or activate a closed session by query, i.e. enter a working state. In a working state, users can store and retrieve persistent objects and create branches for versioning within a session. When users wish to finish collaborative work, or leave a session, they can close the working session temporarily, by entering a closed state. Data associated with the session is still stored. Subsequently, a closed session can be activated to continue the remaining work, i.e. entering a work state again. If the persistent data in a session is not required again, users can delete the session permanently. Users can participate in an interesting session and do collaborative work by looking up the session registry. Finally, administrators of sessions can manage created sessions by using the session handler.

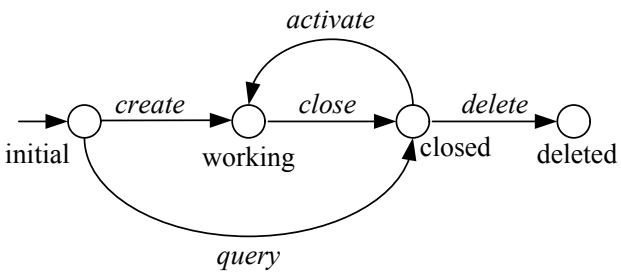

Figure 3. Lifecycle of a persistence session

Through the use of persistence sessions, collaborative applications are capable of supporting synchronous and asynchronous collaboration work. Users joining a working session can work together simultaneously as synchronous collaborations.

Asynchronous users who have their domain knowledge, or are in different positions, can finish their own work items of the workflow by closing and re-activating sessions.

4.3 State Handler

The state handler facilitates the access of application states. A state is composed of a header, which has a unique identifier, a creator, a creation date, a description and a number of items. An item further consists of a header and a body containing a real persistent object, such as an object tree of a run-time state. 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.

Users can delegate any persistent object, encoded in a byte-array or text format, to the state handler. Once the state handler receives the object, it further translates it into a text format and then stores it into the underlying database. For collaborative applications, users can also use the standard Java Serialization APIs to marshal the entire application state as a byte-array object. After serialization, the serialized application state can be saved in the underlying database, via the state handler.

Since Java Serialization APIs do not support anonymous inner classes, users must implement Serializable interface and provide them with class names. Besides, if the collaborative application contains non-serializable objects, like Thread, Socket, IO, etc., default Java Serialization APIs will fail to serialize them and the virtual machine will throw an exception when executions. To solve these problems, Java Serialization APIs allows users to deal with non-serializable objects on their own through a customized function. First, the non-serializable objects must be made transient, which means they will not be handled directly by the virtual machine. Users can employ the customized approach to deal with transient objects for persistence. It furnishes two methods, `writeObject()` and `readObject()`, that allow users to implement customized serializations on their own.

Storing application states sequentially in each persistence session forms a state tree as shown on the right-hand side of Figure 5. The TSPS provides an Indicator in each session to show the current application state. After users store an application state, the Indicator will update from the original state to this last stored state. Figure 4(a) shows the state storing process. When users wish to retrieve a certain state, they can find it by looking up the information contained in the state header and then retrieve it. After retrieval, the Indicator will change to this retrieved state as the current one and the application will recover to this state. Figure 4(b) illustrates the state retrieval process. Users can perform the undo and redo actions by traversing a

state tree. For instance, a developer who wants to build an application can set checkpoints in every development step by storing states. Sometime 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.

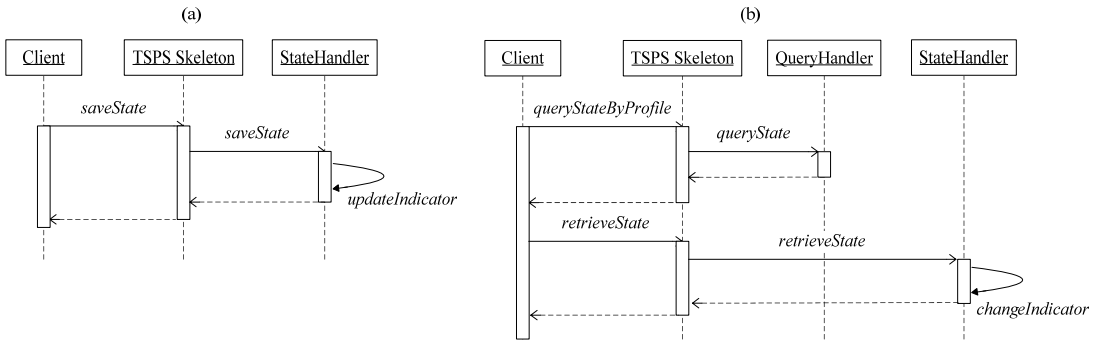

Figure 4. Sequence diagrams for storing a state and retrieving a state

4.4 Version Handler

The 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. In general, a sequence of application states is formed in a single flow without branches, or the latest state replaces the previous one. However, the version handler can arrange states in a tree structure to control versions. Figure 5 shows an example of a state tree. Users can generate branches from existing states for various purposes. We can think of these branches and states as assets that record the development course or execution course of the application. Users can see how the application was implemented or manipulated by tracing the states of each branch. Therefore, when users store states in the TSPS, information about the development or execution is also recorded.

Each branch consists of a unique identifier, a creator, a creation date, and a description. Users can traverse a state tree in each session to retrieve a state using a branch browser. As mentioned above, each session has an Indicator to identify the current work state and show the corresponding branch. Through the branch browser, users can set an Indicator to specify the current work state and branch. There are three types of branch management:

(1) If the Indicator is set in the last state of a branch, i.e. a leaf node, a new stored state is appended to the end of the branch as the latest state and the Indicator is set on this one. Figure 5 shows the state storing process if the

previous state is the last one.

(2) If the indicator is not set in a last state of a branch, i.e. not a leaf node, a new branch is created when a user stores as new state and this state becomes the first node of the branch. Figure 6 shows the state storing process if the previous state is not the last one.

(3) If users wish to create a branch for various purposes, they can do it manually from any existing state. First, it is necessary to create a branch and then switch from the original to the new one. States can then be stored in this new branch. Figure 7 shows the manual creation of a branch.

When users store application states into the TSPS, it will generate a unique identifier in a branch automatically. Through the state identifier, users can specify what state they would like to retrieve. The version handler provides users with an alternative, thereby making state access flexible.

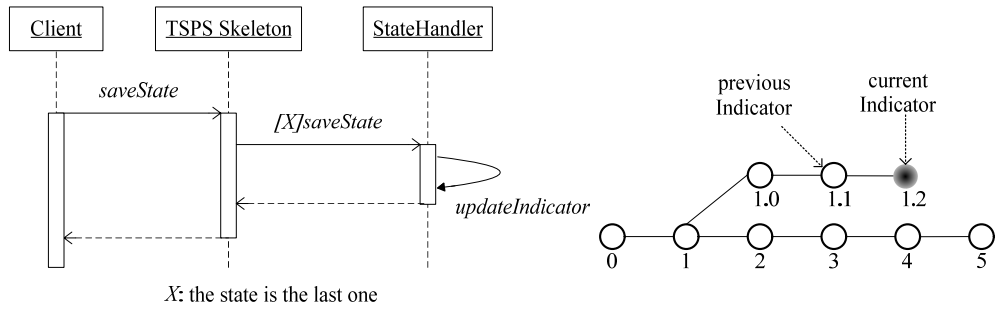

Figure 5. Storage of a state if the previous state is the last one

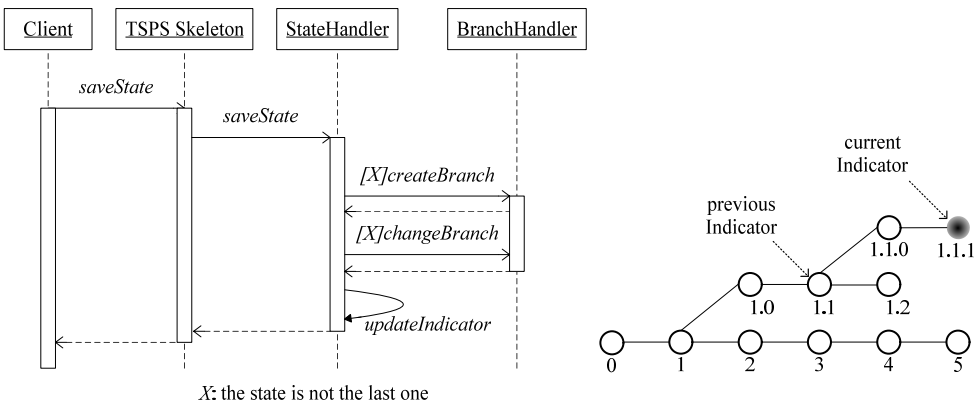

Figure 6. Storage a state if the previous state is not the last one.

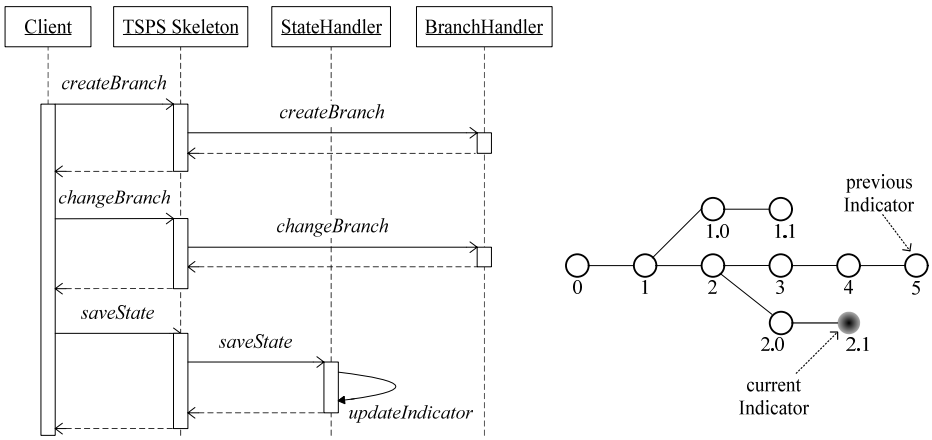

Figure 7. Creating a branch manually.

4.5 Query Handler

The query handler deal with queries related to sessions, states, and branches in the TSPS. For sessions, users can query working sessions by specifying relevant information recorded in session profiles and then join the selected session. Users can also pick a session from a closed list to activate as a live one by specifying queries. For states, users can query states to retrieve by specifying the relevant information recorded in state headers. For branches, users can query branches to switch among them by specifying relevant information recorded in branch profiles. Through the query handler, users can find appropriate sessions, states, and branches.

Since the relevant information about sessions, states, and branches is marked up in a XML manner, we used XPath provided by NXD to perform query actions internally. XPath is used as the query language in the TSPS. It defines a library of standard functions that allows users to write string, number and Boolean expressions to locate where the specified node is in a XML document. We can specify keywords to obtain the query results of sessions, states, and branches by means of XPath.

4.6 Database Accessor

The 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 implementations arbitrarily.

5. Applications of Archiving States Using the Tree-Structured Persistence Server

In this section, we illustrate applications of archiving Java run-time states by employing the TSPS. The first application is the ShareTone Composer as shown in Figure 8. It is built on the ShareTone platform and an integrated development environment for developing collaborative applications in Java. The TSPS is a persistent layer in which ShareTone Composer can store the current states. The ShareTone Composer also works collaboratively itself. Because of the collaborative nature of ShareTone Composer, separate developers can develop Java-based applications cooperatively. After development, these constructed applications are also capable of working collaboratively, so that separate users can finish joint tasks.

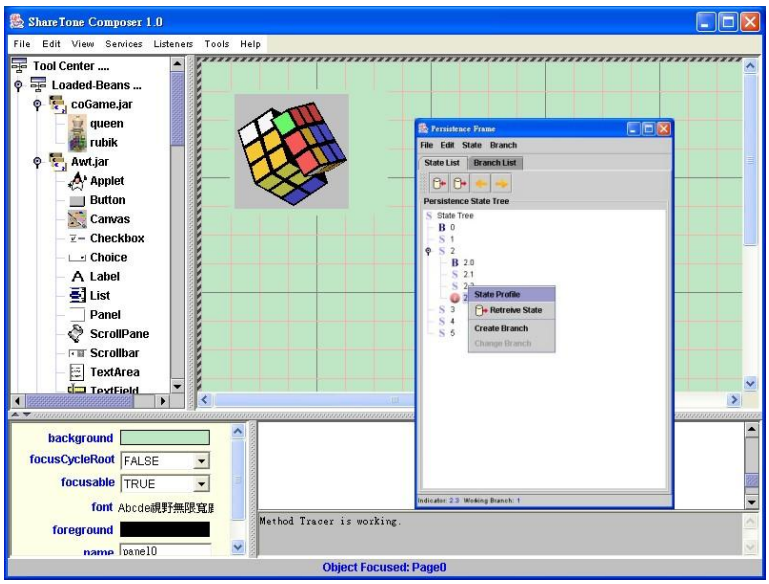

Figure 8. Developing a collaborative application using ShareTone Composer

The ShareTone Composer carries a TSPS driver so that it can access services. When developers start implementing an application, they create a development session first. They can then write and debug their programs of the application by using the ShareTone Composer. During the construction of a collaborative application, developers can save the development states as checkpoints with descriptions. 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. When developers wish to create a new version of the application for another purposes, such as testing or improving, they do not have to leave working session or create a session for a new version. They can just shape a new branch from the original state flow, starting from an existing state, to develop various versions of the application within the same session. Figure 8 shows a front-end interface of a TSPS embedded in ShareTone Composer. At the end of implementation, developers can close the working session. The states and the data associated with the application are kept in the persistence server permanently. Later, developers utilizing the session

query can activate a closed session to restore the last state of the application before closing and continuing the development work.

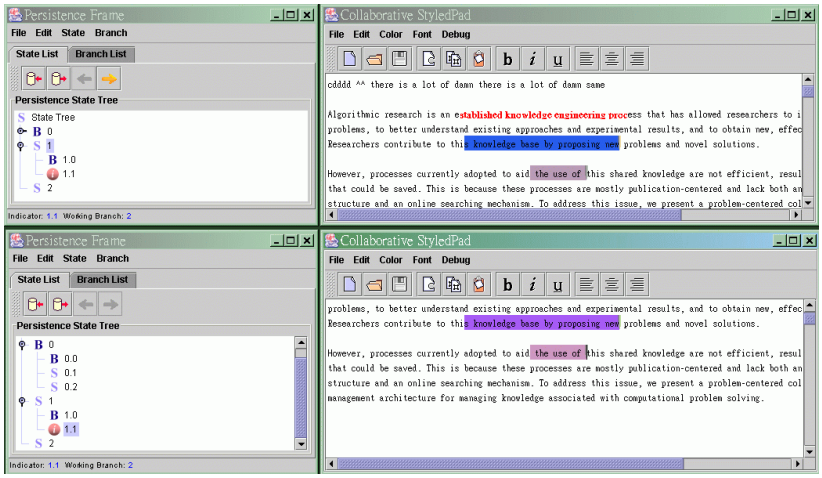

Figure 9. Editing text collaboratively using ShareTone CoEditor

The second application is ShareTone CoEditor, as shown in Figure 9, which allows separate users to simultaneously edit an article together. Users can edit different paragraphs simultaneously in the same article. Also, 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. The ShareTone CoEditor has the persistent capabilities as well as those mentioned in the above paragraph on the TSPS. The ShareTone Composer and the ShareTone CoEditor are two of applications that use the TSPS as a persistence layer to provide persistence services. We implemented the TSPS to save any objects or data that can be encoded in the XML format. Therefore, any applications can use the TSPS to manage persistent objects if these objects can be serialized in XML format.

6. Related Work

Several Java object persistence mechanisms have been proposed to deal with the issues of object persistence. These mechanisms are used in different levels of programming and applied in various domains. The following describes recent Java object persistence technologies.

JDBC (Java Database Connectivity) [33] is a set of common APIs that allow programmers to access databases in Java programming language. Programmers can utilize these common APIs in programs to store data in a database permanently and retrieve data source from a database. Through JDBC APIs, programmers can access a

variety of relational databases, so long as their venders accommodate drivers which implement the JDBC interfaces. Because programmers can use SQL to access databases directly, JDBC APIs allow written database-related programming more flexibility. However, programmers have to spend more time in processing the mapping between objects and tables.

EJB (Enterprise JavaBeans) [34] is introduced in J2EE platform proposed by Sun. There are two types of EJBs, Session Bean and Entity Bean. Entity Bean is an object with properties. Programmers can treat an Entity Bean as a persistence object and need not worry about the low level of database access. Since the container of a J2EE application server would process low-level database access automatically instead of using JDBC APIs, databases are transparent for programmers. Consequently, programmers can concentrate on the business logic of applications by using Entity Beans, rather than write database-related programs.

JDO (Java Data Objects) [35] is a specification developed under the direction of the Java Community Process. It allows users to store and retrieve Java objects transparently by conforming to the defined standard. For Java objects, JDO provides transparent persistence APIs independently from the underlying data stores. The underlying databases could be relational databases or object-oriented databases. The JDO users can use JDOQL to query the database. The JDO and Java Entity Bean users need not be aware of the detail of database access, but they have to create an additional mapping file that describes the relation between object properties and database fields.

JDBC, EJB and JDO are current object persistence technologies that can be used in various levels of persistence data and for different requirements. Since these object persistence mechanisms have their own advantages in different applications and domains, developers can choose a suitable one according to the persistence context. Based on the criteria of encapsulated granularity, simplicity and code intrusion, TSPS and JDO are suitable as a persistence layer in CSCW. Users of a CSCW application are concerned with its states, not with the internal data of each object in a persistence unit. The TSPS encapsulates an application state as a persistence unit. Users can store an application states with its description and subsequently retrieve it to recover by query or traversing the entire state tree. On the one hand, because TSPS doesn’t take account of the internal properties of objects, developers are able to develop persistence-related programs with less code intrusions. On the other hand, the TSPS allows users to manage application states in a tree-structured manner for versioning. In contrast to JDO implementations, the TSPS is a lightweight persistence server, which developers can set up and manipulate with ease.

6. Conclusions and Future Work

As the maturity of Internet, most of applications are derived to become Internet-based ones. Today people are accustomed to employing Internet-based applications, such as e-mail and instant message tools, to work cooperatively. Because of the need for collaborative applications, the idea of computer-supported collaborative work was proposed. By making use of CSCW technologies, people can work together via the Internet more efficiently than before. However, the persistence problem of collaborative applications is a significant issue in the research of computer-supported collaborative work. A collaborative computing environment requires a simple and transparent persistence mechanism to deal with complex data access. Therefore, 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 store 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 TSPSS is not only suitable for CSCW applications. It was 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. Caching mechanisms can enhance performance when users access persistent data via network. Transaction mechanism allows users to start, commit or roll back a transaction. Both mechanisms will make the server more efficient and useful.

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

Reference

[1] C. A. Ellis, S J. Gibbs, and G. L. Rein, “Groupware: Some Issues and Experiences,” Communication of the ACM, vol. 34, no. 1, pp. 38--58, 1991.

[2] Mike Robinson, “Computer Support Co-operative Work: Cases and Concepts,” in proceedings of Groupware'91, Software Engineering Research Centre, Postbus 424, 3500 AK Utrecht, Nederland, pp 59-74, 1991.

[3] L. J. Bannon, nd Kjeld Schmidt, “CSCW: Four Characters in Search of a Context,” in proceedings of the First European Conference on Computer Supported Cooperative Work - ECSCW'91, pp. 3-16, 1991.

[4] W. Reinhard, J. Schweitzer, G. Vlksen and M. Weber, “CSCW Tools: Concepts and Architectures,” IEEE Computer, vol.27, no.5, May 1994

[5] J. Begole, M. Rosson, and C. Shaffer, “Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems,” ACM Transactions on Computer-Human Interaction, 6(2), pp. 95-132. 1999

[6] D. Li, and R. Muntz, “COCA: Collaborative Objects Coordination Architecture,” in proceedings of CSCW'98, pages 179--188, 1998.

[7] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, “Virtual Network Computing,” IEEE Internet Computing, 2(1):33--38, January 1998.

[8] 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 Computer-Supported Cooperative Work, pages 207--217. Association for Computing Machinery, November 1994.

[9] P. Dewan and H. Shen, “Access Control for Collaborative Environments,” in proceedings of CSCW'92, pages 51--58, 1992.

[10] W. Keith Edwards, “Policies and Roles in Collaborative Applications,” in proceedings of CSCW'96, pages 11--20, 1996.

[11] P. Dourish, and V. Bellotti, “Awareness and Coordination in Shared Workspaces,” in proceedings of ACM CSCW Conf., 1992, 107-114.

[12] C. Gutwin, R. Roseman, and S. Greenberg, “A usability study of awareness widgets in a shared workspace groupware system,” in proceedings of ACM Conference on Computer Supported Cooperative Work (CSCW'96), Boston, Mass., pp.258-267, 1996.

[13] J. A. Mariani, and T. Rodden, “The Impact of CSCW on Database Technology,” in Proceedings of IFIP International Workshop on CSCW, Berlin, Germany, pp. 146-161, 1991.

[14] A. Prakash, H. S. Shim, and J. H. Lee, “Data Management Issues and Trade-Offs in CSCW Systems,” IEEE Transaction on Knowledge Data Engineering 11(1): 213-227, 1999.

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

[16] 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 5, 2, pp. 147–167, 1987.

[17] P. Cederqvist et al. Version management with CVS, 1992, available at http://www.cvshome.org/

[18] IBM Rational ClearCase, available at http://www-306.ibm.com/software/awdtools/clearcase/

[19] Microsoft Visual SourceSafe, available at http://msdn.microsoft.com/ssafe/

[20] James Gosling and Bill Joy, "The Java Language Specification, 2th ed," Addison-Wesley, 1996.

[21] Java Serializable APIs, available at http://java.sun.com/j2se/1.4.1/docs/api/java/io/Serializable.html

[22] Long-Term Persistence for JavaBeans, available at http://java.sun.com/products/jfc/tsc/articles/persistence/

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

[24] Koala Bean Markup Language: KBML enables one to serialize/deserialize JavaBeans™ to/from XML documents, available at http://www-sop.inria.fr/koala/kbml/

[25] 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, pages 223-40, Kyoto, Japan, December 1989.

[26] Mansour Zand, Val Collins, Dale Caviness, "A survey of current object-oriented databases," ACM SIGMIS Database, Volume 26 Issue 1, February 1995.

[27] XML:DB Initiative, available at http://www.xmldb.org/

[28] Kimbro Staken, Introduction to Native XML Databases, available at http://www.xml.com/pub/a/2001/10/31/nativexmldb.html

[29] W3C. XML Path Language (XPath). Working Draft, November 1999, available at http://www.w3.org/TR/xpath.

[30] Apache Xindice, available at http://xml.apache.org/xindice/

[31] Wolfgang Meier. eXist, "An Open Source Native XML Database," In: Akmal B. Chaudri, Mario Jeckle, Erhard Rahm, Rainer Unland (Eds.): Web, Web-Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops, Erfurt, Germany, October 2002, available at http://exist.sourceforge.net/

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

[33] JDBC Technology, available at http://java.sun.com/products/jdbc/index.html

[34] Enterprise JavaBeans Technology, available at http://java.sun.com/products/ejb/

[35] Java Data Objects, available at http://java.sun.com/products/jdo/