An Integrated Session and Repository Management Approach for Real-Time Collaborative Editing Systems

Steven Xia
Griffith University
Brisbane, Australia
Q.Xia@griffith.edu.au

David Sun
University of California at Berkeley
Berkeley, CA, USA
DavidSun@berkeley.edu

Chengzheng Sun
Nanyang Technological University
Singapore 639798
CZSun@ntu.edu.sg

David Chen
Griffith University
Brisbane, Australia
D.Chen@griffith.edu.au

Abstract

Real-time collaborative editing systems may be applied in collaborative activities with a quite spontaneous and impromptu style. In such application activities, the session awareness information is particularly important. One key technical issue in achieving this flexibility is how collaboration sessions are managed in the system. However, existing approaches to session management are unable to provide sufficient session awareness information. In this paper, we propose a novel Integrated Repository and Session Management (IRSM) approach for supporting real-time collaborative editing. IRSM supports lightweight session management to significantly reduce the overhead of explicit session initialization; and detailed session awareness information through the integration of session and repository management mechanisms. In addition, IRSM is application-independent, so it can be used to support a wide range of collaborative editing systems. The feasibility and generality of this approach has been demonstrated in two real-time collaborative editing systems: CoWord and CoPowerPoint.

1. Introduction

Real-time collaborative editing systems are a useful and interesting type of distributed systems. They support geographically distributed users to collaboratively edit shared documents at any time over the internet without imposing unnecessary constraints on the users. Existing real-time collaborative editing systems have been designed to handle a variety of documents, which can be as simple as plain text files [13] or as complex as word processing documents [14].

Real-time collaborative editing systems may be used in a range of collaborative activities including collaborative document authoring, meeting discussion recording and brainstorming, which demonstrate the following characteristics. First, they may adopt different collaboration styles, ranging from non-real-time collaboration (e.g. co-authors write separate chapters of a paper individually) to real-time collaboration (e.g. co-authors discuss to determine some parts of the paper in an online session). Moreover, even a single collaborative activity may need multiple collaboration styles. For example, while co-author A is editing the shared document in a non-real-time mode, co-authors B gets online and joins the collaboratively editing session. In this case, this collaborative activity switches to a real-time mode. Consequently, the dynamic membership is another characteristic of collaborative document authoring since co-authors may join or leave a collaborative editing session at any time.

These characteristics also appear in other application areas of real-time collaborative editing systems like formal meetings. For example, a meeting does not have to wait for all attendees to be present to start. Latecomers may join the ongoing meeting at any time. Also, attendees may quit a meeting at any time, but the meeting does not have to terminate.

Session management plays a key role in collaborative systems. It determines how collaboration sessions are initiated and terminated and how individuals join and leave a session [11]. The spontaneous and impromptu collaboration style of real-time collaborative editing systems requires the support from flexible and lightweight session management approaches.

In literature, this kind of spontaneous and impromptu collaboration style is supported by implicit session management [4] approaches. These approaches make use of users' object-accessing actions to trigger collaboration session management actions. For example, users who have opened the same document are automatically placed in the same collaborative editing session by the session manager. In this way, session management becomes transparent to users.

However, existing implicit session management approaches cannot provide users with sufficient session awareness information that describes which document is being edited in a session, which users are participating in the session and attributes of participating users etc. This situation introduces the following negative implications to the usability of the system. First, it is the system rather than users who determines whether to collaborate [8]. The system's decision may violate users' intentions. On the other hand, Users do not have enough knowledge to predict which session-related events will be triggered by their document accessing actions. They do not know whether they will be thrown into a session by opening a document. Second, it eliminates the possibility for users to control the session and to perform session-related actions. Many of these actions are essential elements of the collaboration process itself. In a formal meeting setting, for instance, session-management activities which should be placed under the discretion of the meeting host include: starting/ending the meeting, inviting additional participants, and excluding uninvited participants.

We address this issue in this paper by introducing a novel *Integrated Repository and Session Management* (IRSM) approach for supporting collaborative editing systems in this paper. On the one hand, our approach manages collaborative editing sessions based on user actions (e.g. open or close documents) thus to avoid the overhead of explicit session management actions and support the flexible collaboration styles. On the other hand, our approach integrates session and repository management mechanisms, so that it can provide users with integrated session and repository information. With this knowledge, users can establish a relationship between shared documents and collaborative editing sessions and make their own decisions on collaborative actions. Moreover, our approach is independent of any concrete applications, so it can be used to develop session management systems that are capable of supporting a wide range of collaborative editing systems.

The rest of this paper is organized as following. First, related session management approaches are reviewed in Section 2. Next, the IRSM document repository management mechanism is discussed in Section 3. In Section 4, we discuss the IRSM session management mechanism. In Section 5, techniques for processing the integrated session and repository information are presented. Finally, contribution and future work are summarized in Section 6.

2. Related Work

Existing session management approaches fall into two categories [4], which are explicit and implicit session management approaches.

2.1. Explicit Session Management Approaches

Explicit session management approaches initially appeared in some early teleconferencing systems and are still widely used today in varieties of systems. The major characteristic of these approaches is that they require users to take explicit actions to initiate a collaboration session. Some of them require an initiating user to invite others into a session, while others require users to find an existing collaboration session and join.

MMConf [3] adopts an *initiator-based* approach. After the initiating user creates a session, invitations are sent to other users, who will select to accept or reject the invitations. The session begins after all invited users have responded or time out.

Collage[1] adopts a *joiner-based* approach. To join an existing session, a user needs to manually input the IP address or machine name of the session host and the port number of the session process. Similarly, Flexible JAMM [1] also adopts this approach.

Explicit session management approaches are suitable for supporting formal collaboration because they facilitate explicit session-related actions. However, they are not suitable for the lightweight session management of collaborative editing systems, mainly because explicit inviting or joining actions involve too much overhead, which drastically decreases the system usability. Furthermore, these approaches nearly provide no session awareness information.

2.2. Implicit Session Management Approaches

Implicit session management approaches are designed to avoid the overhead of explicit session management approaches. These approaches manage collaboration sessions based on users' actions in the collaborative environment. According to the types of user actions they utilize, implicit session management approaches are classified as *artifact-based*, *place-based* and *activity-based* approaches.

With artifact-based approaches, users accessing the same artifact are joined in the same session. One

[1] NCSA Collage: http://archive.ncsa.uiuc.edu/SDG/Software/XCollage/collage.html. Accessed 18 Aug. 2005.

typical example is Intermezzo [4]. In the Intermezzo system, applications publish activity records including identifiers of the user, application and object to the session manager. The session manager searches for records with overlapping object identifiers. If such records are found, the session manager sends events to the corresponding applications to notify them about the collaboration potential. Then the applications are responsible for initiating the collaboration on their own.

Place-based approaches join users who have entered the same place in the same session. These approaches are frequently adopted in virtual space systems such as MASSIVE [7]. In MASSIVE, users are represented as objects equipped with communication media in a virtual 3D world. When two objects are approximate enough and they support common communication media, a peer connection is established between them.

Rusken [12] adopts a hybrid of artifact-based and place-based approach. In the artifact-based aspect, it extends the concept of *object* in implicit session management to *a set of objects*. Users accessing the same object set are joined in the same session. In place-based aspect, session joining and leaving are triggered by events of users entering and leaving locations.

An activity-based approach is adopted in Piazza [9]. The Encounter tool of the Piazza system detects users who are performing the same task (e.g. viewing the same web page, editing the same document) and creates connections for them to communicate.

Collaborative editing is a document-centric activity, so artifact-based approaches seem naturally applicable. However, existing artifact-based implicit session management approaches (e.g. Intermezzo) are not suitable for supporting real-time collaborative editing systems because they could not provide users with sufficient session awareness information.

The reason for the lack of session awareness information is that these approaches assume that users explicitly interact with a document repository to access shared documents and implicitly interact with the session manager. However, since the session manager is transparent to users, it cannot provide session awareness information to users. Being separated from the session manager, the document repository manager cannot provide users with the session awareness information either.

Therefore, the root of this problem is the separation of session management and repository management mechanisms. To solve this problem, we need to integrate these two mechanisms so that we can provide users with integrated session and repository information. With this knowledge users can establish the relationship between documents stored in the repository and ongoing collaboration sessions. Moreover, the integrated information can also facilitate users to perform explicit session-related actions.

The IRSM approach consists of three technical elements, which are the document repository management mechanism, collaborative editing session management mechanism and the integrated session and repository information processing mechanism. In the following sections, we shall discuss details of these mechanisms to illustrate how the above desirable effects are achieved.

3. The IRSM Repository Management

The main purpose of the IRSM document repository management mechanism is to provide shared document storage and access services to collaborative editors.

The organization of shared documents in a document repository is shown in Figure 1. A document repository consists of a series of sub-repositories (shown in Figure 1-(a)). Each sub-repository contains a tree structure of directories and documents (shown in Figure 1-(b)). Users can customize the structure of sub-repositories by adding or deleting documents and directories.

Sub-repositories are protected by user accounts and passwords. User accounts and sub-repositories are associated in a many-to-many relationship. The system can be dynamically configured to determine which sub-repositories a user account is associated with. When a user logs in the repository with an account, sub-repositories associated with the account are displayed, while others are inaccessible. Multiple users may have access to common sub-repositories. From these common sub-repositories, users may initiate collaborative editing sessions.

Documents stored in the repository are identified

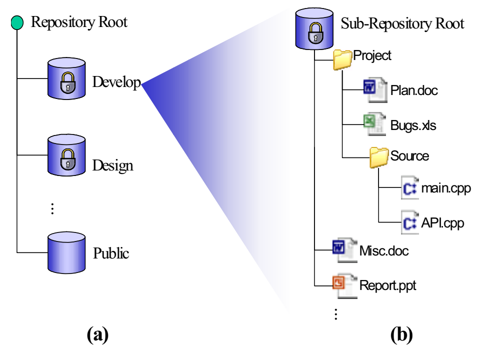


Figure 1. The tree structure of the document repository. (a) Repository root and sub-repositories, and (b) the tree structure of a sub-repository.

with the sub-repository name and the full path name in the sub-repository. With this identifier, documents can be loaded and saved. For example, to access the document "Plan.doc" in Figure 1-(b), a *LoadDocument* request containing the document identifier "Develop: /Project/Design.doc" is sent to the repository manager. Then the repository manager authenticates the request, reads that document and sends the document content back to the requester. Furthermore, to save this document to the sub-repository, a *SaveDocument* request containing the document content and document identifier is sent to the repository manager, which will save the document content to the specified path.

The major advantage of using account-protected sub-repositories to organize shared documents is that it facilitates formal collaboration. For instance, a formal meeting requires a fixed attendee group. To support such a meeting, we can create a sub-repository and assign access rights to all invited users. So, only invited users (i.e. those who are informed of the account name and password) are allowed to join the meeting and uninvited users are excluded.

To support publicly shared document storages that allow any user to access, we can create sub-repositories that are not protected by user accounts (e.g. the "Public" sub-repository shown in Figure 1-(b)).

4. The IRSM Session Management

4.1. The Implicit Session Management

The architecture of the IRSM session management is shown in Figure 2.

As shown in Figure 2-(b), an IRSM session manager is capable of handling multiple sessions at the

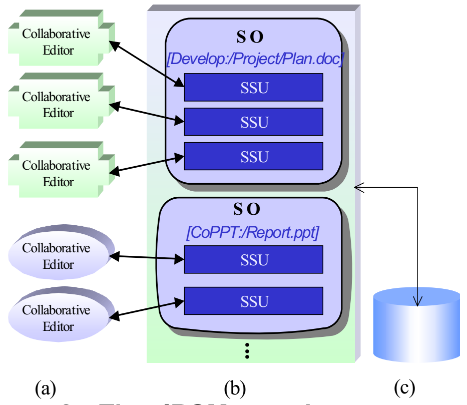


Figure 2. The IRSM session management architecture. (a) The collaborative editors; (b) the session manager; and (c) the repository manager.

same time. Session information is contained in *Session Objects* (SO, represented as the rounded rectangles in Figure 2-(b)). An SO is identified with the identifier of the document that is being collaboratively edited in this session. Each SO contains multiple *Session Service Units* (SSU). An SSU is a thread that maintains a network connection to a collaborative editor and provides session management services to the connected collaborative editor.

Figure 2 also illustrates the integration of session management and repository management. Although the session manager encapsulates session management functionalities, it provides document-accessing interfaces to collaborative editors. On the other hand, collaborative editors do not have direct access to the repository manager, so document-accessing requests are sent to the session manager.

This session management approach is able to achieve the effects of implicit session management. To perform session management tasks, collaborative editors only need to send simple document-accessing requests to the session manager, which include *LoadDocument* and *CloseDocument*.

After receiving a *LoadDocument* request, the session manager checks whether there is an existing SO for the requested document. If not, the session manager performs a *session creation* process in the following steps.

1. The session manager creates a new SO with the identifier of the requested document.
2. The session manager creates an SSU in the newly created SO to provide services to the requesting collaborative editor.
3. The session manager forwards the request to the repository manager to obtain the requested document.
4. The session manager sends the document content back to the requesting collaborative editor.

If such an SO is found, then there must be an existing session in *which* users are collaboratively editing the requested document. In this case, the session manager performs a *session joining* process to join the requesting site into this session.

1. The session manager creates a new SSU thread in this SO to provide services to the joining collaborative editor.
2. The session manager initiates a late-joining process (to be discussed in Section 4.2) to join the requesting site into this session.

After receiving a *CloseDocument* request, the session manager disconnects the requesting collaborative editor and deletes its SSU. If this SO contains no more SSU, the SO is also deleted.

Collaborative editors can also save a document to the repository by sending a *SaveDocument* request. After receiving this request, the session manager simply forwards the request to the repository manager, which shall write the document content attached in the request into the specified document in the repository.

4.2. Accommodating Late Joiners

Accommodating later joiners is an essential session management functionality for supporting the spontaneous and impromptu collaboration style of real-time collaborative editing systems. With the support of this functionality, new collaborating sites can smoothly join ongoing collaboration sessions at any time.

For the late joining site to start collaboration with existing sites, its states relevant to the collaboration should be consistent with existing ones. For the convenience of discussion, we refer to these states as *shared states* in the sense that these states are shared by all sites in the same session. Therefore, a late joining mechanism needs to have capabilities to (1) identify the shard states, (2) extract the shared states from existing sites, (3) import the shared states to the joining site, and (4) create an appropriate setting to perform the shared states extraction and importing.

4.3.1. Shared State Identification, Extraction and Importing. The most widely adopted approaches to shared state extraction and importing are application-transparent approaches. Without the capabilities of identifying which pieces of state information of the underlying application are relevant to the collaboration and manipulating such information, these approaches strive to make every detail of the joining site identical to existing sites.

Application-transparent late joining approaches fall into three categories, which are (1) *event replay*, which records all input events to an existing site and replays the events to the *joining* site, (2) *image copy* [2], which copies the process image in the memory of an existing site and import the image to the joining site, and (3) *runtime component replacement* [1], which use selected application components of an existing site to replace counterparts of the joining sites.

Although these approaches have the advantage of being completely application-independent, they suffer from different problems. For example, the event replay approach is inefficient, while image copy and runtime component replacement approaches are not widely supported by underlying platforms.

The application semantic knowledge (e.g. what data objects the application manipulates, and what functionalities the application provides, etc.) is the key to solving these problems in our approach. With the application semantic knowledge, we can precisely identify the shared states that should be transferred to the joining site.

A collaborative editor has numerous pieces of state information, such as the document content, the cursor position and the window size, etc. The shared states usually cover only a subset. However, this subset varies according to the functionalities of the collaborative editors and the collaboration styles. For example, for an unconstrained real-time collaborative editor aiming only to keep the consistency among shared document copies, the shared states only cover the shared document content and the editing operation history. If the collaborative editing system supports telepointers [6], then mouse cursor positions of all existing sites should also be included in the shared states.

Furthermore, with the application semantic knowledge, we can invoke the application functionalities to perform the shared state extraction and importing. For example, to extract and import the shared document content, we can invoke the document saving and loading functionalities of the collaborative editor. In other words, to support our late joining approach, the collaborative editors need to provide corresponding functionalities.

4.3.2. The Late Joining Protocol. Creating an appropriate setting for the late joining process is also important. This is because in real-time collaborative editing systems, shared states of each site are in dynamic changes in the face of concurrent editing operations. At any moment, collaborating sites may have temporarily divergent shared states. This brings two problems to the late joining mechanism. First, among the existing sites with divergent shared states, which site has the proper shared states to extract? Second, concurrent editing operations may interleave with the shared states extraction and importing. After the joining site has imported the shared states, how can we guarantee that the shared states are still consistent to those of the extracting site?

Our solution to these problems is to force all existing sites to enter a quiescent state, in which no sites are generating editing operations and no editing operations are being transmitted in the communication channel. In a quiescent state, all existing sites should have consistent shared states, so any one of them can be selected to extract the shared states. The quiescent state ends after the joining process is finished. This effectively prevents concurrent editing operations from destroying the consistency between the joining site and existing sites.

Our late joining protocol is a session manager-coordinated checkpoint-based protocol, which involves the following steps.

1. The session manager broadcasts a *CheckPointBegin* message to all existing sites in the session.
2. When an existing site receives this message, it blocks the input from the local user. This prevents the generation of editing operations.
3. Then it sends back a *CheckPointReady* message to the session manager as a response.
4. After all responses have been received, the session manager randomly chooses an existing site and sends it a *SharedStateRequest* message.
5. When the chosen site receives this request, it extracts the shared states and sends it back to the session manager.
6. The session manager sends the shared states to the joining site in a *JoinApproval* message.
7. The session manager broadcasts a *CheckPointFinish* message to all sites.
8. Upon receiving the *CheckPointFinish* message, all sites unblock the local input and allow the collaboration to continue.

When the session manager receives *CheckPointReady* messages from all existing sites (step 4), no more editing operations will be sent to it. This is because each site is connected to the session manager with a single communication channel. In this channel, message orders are naturally preserved. For the same reason, when the chosen site receives the *SharedStateRequest* message (step 4), no more editing operations will be sent to this site during the joining process. At this moment, this site reaches a quiescent state and the stable shared states can be extracted from this site.

After the joining site receives the shared states (step 7), it shares the consistent state with the site that has exported the shared states. Furthermore, they are both in the quiescent state.

Finally, when all sites have received the *CheckPointFinish* message (step 8), they are all in the quiescent state and their shared states are consistent. Based on the consistent shared states, they can resume the collaboration.

5. Processing the Integrated Session and Repository Information

Based on the integrated session and repository management mechanisms, we are able to provide users with the integrated information of both the document repository and the collaborative editing sessions. In this section, we shall discuss issues related to the integrated information processing.

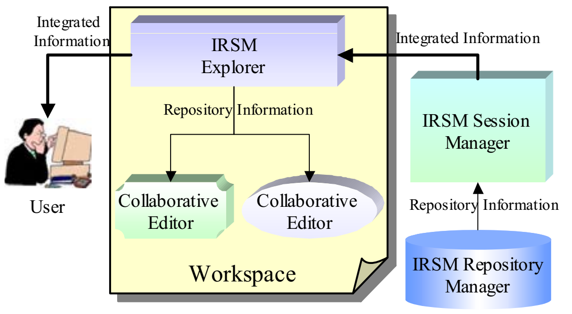


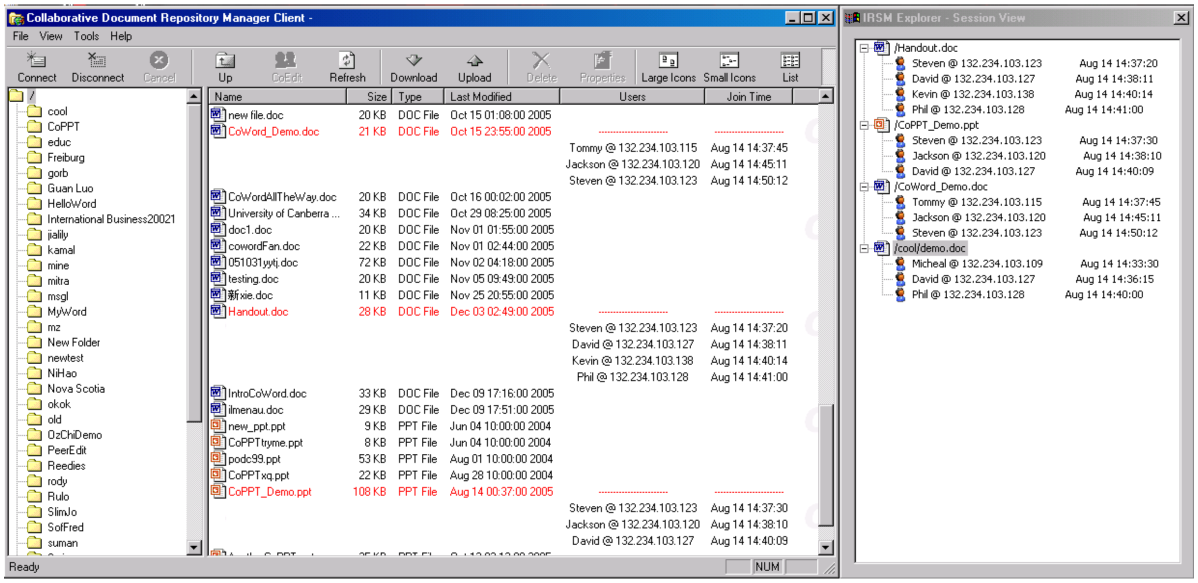
Figure 3. The repository and session information flow in the IRSM system.

We have developed an integrated repository and session management system, the IRSM system. As shown in Figure 3, this system contains three components: the IRSM Session Manager, the IRSM Repository Manager and the IRSM Explorer. The IRSM Session Manager and Repository Manager perform session and repository managing tasks as discussed in this paper. The IRSM Explorer provides the following functionalities: (1) interpreting and representing the integrated session and repository information to users; (2) supporting users to access documents stored in the repository; (3) supporting users to customize the structure (e.g. creating or deleting directories, downloading or uploading documents) of sub-repositories; and (4) supporting users to take explicit session-related actions. In this section, we shall use the IRSM system as an example to illustrate techniques to process the integrated information.

The IRSM Explorer runs in the user's local system. Furthermore, to support collaborative editing on shared documents, a collection of collaborative editors need to be installed in the user's local system. All these applications comprise the user's workspace (as shown in Figure 3).

As shown in Figure 3, the session and repository information flows from the server end (the session manager and repository manager) to the user's workspace.

The information flow starts from the IRSM Repository Manager. It provides repository information to the IRSM Session Manager. This information contains the directory structure and document attributes of sub-repositories. The IRSM Session Manager generates the integrated session and repository information by adding session information to the repository information. The integrated information is sent to and interpreted by the IRSM



/
cool
CoPPT
educ
Freiburg
gorb
Guan Luo
HelloWord
International Business20021
jialily
kamal
mine
mitra
msgl
MyWord
mz
New Folder
newtest
NiHao
Nova Scotia
okok
old
OzChiDemo
PeerEdit
Reedies
rody
Rulo
SlimJo
SofFred
suman

Name	Size	Type	Last Modified	Users	Join Time
new file.doc	20 KB	DOC File	Oct 15 01:08:00 2005		
CoWord_Demo.doc	21 KB	DOC File	Oct 15 23:55:00 2005	--------------------	--------------------
				Tommy @ 132.234.103.115	Aug 14 14:37:45
				Jackson @ 132.234.103.120	Aug 14 14:45:11
				Steven @ 132.234.103.123	Aug 14 14:50:12
CoWordAllTheWay.doc	20 KB	DOC File	Oct 16 00:02:00 2005		
University of Canberra ...	34 KB	DOC File	Oct 29 08:25:00 2005		
doc1.doc	20 KB	DOC File	Nov 01 01:55:00 2005		
cowordFan.doc	22 KB	DOC File	Nov 01 02:44:00 2005		
051031yytj.doc	72 KB	DOC File	Nov 02 04:18:00 2005		
testing.doc	20 KB	DOC File	Nov 05 09:49:00 2005		
新xie.doc	11 KB	DOC File	Nov 25 20:55:00 2005		
Handout.doc	28 KB	DOC File	Dec 03 02:49:00 2005	--------------------	--------------------
				Steven @ 132.234.103.123	Aug 14 14:37:20
				David @ 132.234.103.127	Aug 14 14:38:11
				Kevin @ 132.234.103.138	Aug 14 14:40:14
				Phil @ 132.234.103.128	Aug 14 14:41:00
IntroCoWord.doc	33 KB	DOC File	Dec 09 17:16:00 2005		
ilmenau.doc	29 KB	DOC File	Dec 09 17:51:00 2005		
new_ppt.ppt	9 KB	PPT File	Jun 04 10:00:00 2004		
CoPPTtryme.ppt	8 KB	PPT File	Jun 04 10:00:00 2004		
podc99.ppt	53 KB	PPT File	Aug 01 10:00:00 2004		
CoPPTxq.ppt	22 KB	PPT File	Aug 28 10:00:00 2004		
CoPPT_Demo.ppt	108 KB	PPT File	Aug 14 00:37:00 2005	--------------------	--------------------
				Steven @ 132.234.103.123	Aug 14 14:37:30
				Jackson @ 132.234.103.120	Aug 14 14:38:10
				David @ 132.234.103.127	Aug 14 14:40:09

/Handout.doc
- Steven @ 132.234.103.123 Aug 14 14:37:20
- David @ 132.234.103.127 Aug 14 14:38:11
- Kevin @ 132.234.103.138 Aug 14 14:40:14
- Phil @ 132.234.103.128 Aug 14 14:41:00

/CoPPT_Demo.ppt
- Steven @ 132.234.103.123 Aug 14 14:37:30
- Jackson @ 132.234.103.120 Aug 14 14:38:10
- David @ 132.234.103.127 Aug 14 14:40:09

/CoWord_Demo.doc
- Tommy @ 132.234.103.115 Aug 14 14:37:45
- Jackson @ 132.234.103.120 Aug 14 14:45:11
- Steven @ 132.234.103.123 Aug 14 14:50:12

/cool/demo.doc
- Micheal @ 132.234.103.109 Aug 14 14:33:30
- David @ 132.234.103.127 Aug 14 14:36:15
- Phil @ 132.234.103.128 Aug 14 14:40:00

Figure 4. The Integrated information representation in the IRSM Explorer, (a) the Repository View; and (b) the Session View.

Explorer and displayed in its user interface. Users obtain the integrated information from the IRSM Explorer and evaluate this information before issuing document-accessing commands. Finally, while starting up corresponding collaborative editors, the IRSM Explorer passes them the selected document path information, which is a fraction of the repository information.

The integrated session and repository information is provided to users to enhance session awareness. At the same time, this information is also used to facilitate explicit session-related actions that are supported in explicit session management approaches. It requires that the integrated session and repository information be effectively represented to achieve these purposes. In the IRSM Explorer, we represent the integrated information in two views as shown in Figure 4.

The first view is the *Repository View* (shown in Figure 4-(a)). This view represents tree-like structures of the accessible sub-repositories (including directories and documents) in a Windows Explorer-like user interface. The directory structure and documents in the current directory are displayed in two panes respectively. In this view, users can directly and conveniently browse the directory structure of sub-repositories (see the left pane of Figure 4-(a)) and documents with attributes in the current directory (see the right pane of Figure 4-(a)). Users can also directly issue document-accessing commands in this view (e.g. by double-clicking a document item, or clicking the *CoEdit* item in the toolbar or menu), which may trigger session-related actions. As shown in Figure 4-(a), the session awareness information is represented as an attribute of documents and is attached to individual documents. Unlike other document attributes (e.g. size, type, created time, etc.), a session attribute of a document is a nested one, which contains attributes in further levels. Information contained in a session attribute includes properties of users participating in this session. A user has a series of attributes such as user name, IP address, joining time, etc. This view solves session awareness information problem of implicit session management approaches. With this view, users can predict consequences of their actions before accessing a document by evaluating the session awareness information.

The second view is the Session View (Figure 4-(b)). This view provides a user interface that only lists all sessions in the accessible sub-repositories. Sessions are represented in a tree structure. Collaborative editing sessions are listed as the top-level nodes. Each session node is named after the document edited in this session. Users in a session with their attributes are listed as child nodes subordinate to the session node. This view solves the difficulty of supporting explicit session-related actions. With this interface, users can browse all existing sessions with a glance. To join an existing session, the user only needs to select the session node and double-click it with the mouse.

6. Conclusion and Future Work

In this paper, we have presented an Integrated Repository and Session Management approach for supporting collaborative editing systems. This approach not only supports the spontaneous and impromptu collaboration styles of collaborative editing systems, but also provides users with integrated session and repository information so that they can make their own decisions on their collaborative actions and explicitly perform session-related actions. Moreover, our approach is generic so that we can develop application-independent session and repository management systems to support a wide range of collaborative editors.

Based on the IRSM approach, we have developed a reusable session and repository management system, called IRSM system. Currently, we are using this system to support collaborative editing sessions for CoWord (http://reduce.qpsf.edu.au/coword), which is a collaborative Word processor, and CoPowerPoint (http://reduce.qpsf.edu.au/copowerpoint), which is a collaborative slide authoring and presentation system. Currently, we are on the way developing other collaborative editors (e.g. collaborative CAD and web design systems). We shall also support these systems with the IRSM system to further verify the generality of our approach and to investigate session management requirements of different collaborative editing systems. Moreover, we plan to design more comprehensive session awareness features to improve the efficiency of collaboration.

7. References

[1] Begole, J., Rosson, M., and Shaffer, C. "Flexible collaboration transparence: supporting worker independence in replicated application-sharing systems". *ACM Transactions on Computer-Human Interaction*, 1999, 6(2), pp. 95 – 132.

[2] Chung, G. and Dewan, P. "A mechanism for supporting client migration in a shared window system". *Proc. ACM Symposium on User Interface Software and* Technology, 1996, pp. 11–20.

[3] Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. "MMConf: an infrastructure for building shared multimedia application". *Proc. ACM Conference on Computer-Supported Cooperative Work*, 1990, pp. 329 – 342.

[4] Edwards, W., K. "Session management for collaborative applications". *Proc. ACM Conference on Computer-Supported Cooperative Work*, 1994, pp. 323 – 330.

[5] Ellis, C. A. and Gibbs, S. J. "Concurrency control in groupware systems". *Proc. the ACM Conference on Management of Data*, 1989, pp. 399-407.

[6] Greenberg, S., Gutwin, C., and Roseman, M. "Semantic telepointers for groupwares". *Proc. Australian Conference on Computer-Human Interaction*, 1996, pp. 54 – 61.

[7] Greenhalgh, C. and Benford, S. "MASSIVE: a collaborative virtual environment for teleconferencing". *ACM Transaction on Computer-Human* Interaction, 1995, (2)3, pp. 239-261.

[8] Gutwin, C., Greenberg, S., Blum, R., and Dyck, J. "Supporting informal collaboration in shared-workspace groupware". *HCI Technical Report 2005-01.*

[9] Isaacs, E., A., Tang, J., C., and Morris, T. "Piazza: a desktop environment supporting impromptu and planned interactions". *Proc. ACM Conference on Computer-Supported Cooperative Work*, 1996, pp. 315 – 324.

[10] Kanawati, R. "LICRA: a replicated-data management algorithm for distributed synchronous groupware application". *Journal of Parallel Computing*, 1997, 22, pp. 1733 – 1746.

[11] Patterson, J. F., Hill, R. D., Rohall, S. L., and Meeks, S. W. "Rendezvous: an architecture for synchronous multi-user applications". *Proc. ACM Conference on Computer-Supported Cooperative Work,* 1990, pp. 317-328.

[12] Texier, G. and Plouzeau, N. "Automatic Management of Sessions in Shared Spaces". *Journal of Super Computing*, 2003, 24(2), pp.173-181.

[13] Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. "Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems". *ACM Trans. on Computer-Human Interaction*, 1998, 5(1), pp. 63-108.

[14] Xia, S., Sun, C., Sun, D., Shen, H., and Chen, D. "Leveraging single-user applications for multi-user collaboration: the coword approach". *In Proc. of ACM Conference on Compute- Supported Cooperative Work*, 2004, pp. 162–171.