LEVERAGING SINGLE-USER APPLICATIONS FOR MULTI-USER COLLABORATION

By

Qian Xia

A DISSERTATION SUBMITTED IN FULFILLMENT OF THE REQUIREMENT OF
THE DEGREE OF
DOCTOR OF PHILOSOPHY

AT
THE SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
FACULTY OF ENGINEERING AND INFORMATION TECHNOLOGY
GRIFFITH UNIVERSITY
BRISBANE, QLD 4111, AUSTRALIA
MARCH 2006

GRIFFITH UNIVERSITY

Date: March 2006

Author: Qian Xia

Title: Leveraging Single-User Applications for Multi-User Collaboration

Department: School of Information and Communication Technology

Degree: Ph.D. Year: 2006

I declare this work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

_____

Signature of Author

Table of Content

List of Figures

List of Tables

Acknowledgement

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Chengzheng Sun. He has looked after me throughout my PhD study period. He offered enlightening and insightful ideas in my research project, offered critical but helpful comments on my results, offered encouragement in times of difficulties, and offered valuable advice in my living. More importantly, his spirit in hardworking and pursuing academic excellence has significantly affected my attitude to my study and work. What I have leaned from him has not only helped me complete my PhD study, but will also help me in my academic career in the future.

I would also like to thank my supervisor Dr. David Chen. He is an important mentor and friend, who has helped me in many aspects, including providing inspiring ideas for my research, helping me revise papers and thesis and arrange my academic activities. Without his help, I should have encountered many more difficulties. I would also like to thank my associated supervisor, Prof. Geoff Dromey, who has also offered me generous help.

Furthermore, I would like to thank colleagues in my groups, including David Sun, Jingzhi Guo, Haifeng Shen and Kevin Lin. I have gained a lot through cooperative work and discussion with them.

No words could express how much I am indebted to my family. My parents have done everything they could to help me accomplish my study. My father, who sadly passed away in the last stage of my PhD study, did not see me gaining the degree. I believe my completing this thesis is the best gift for him. I also owe a lot to my wife. She accompanied me in these years, perfectly arranging my everyday living, which allowed me to focus on my study and research. Without her strong support and passionate love, I could not have finished this thesis.

Publications Derived from This Research

1. Sun, C., Xia, S., Sun, D., Chen, D., Shen, H. and Cai, W. “Transparent adaptation of single-user applications for multi-user real-time collaboration”. To appear in *ACM Transactions in Human – Computer Interaction*.
2. Xia, S., Sun, D., Sun, C. and Chen, D. 2006. “An integrated session and repository management approach for real-time collaborative editing systems”. In *Proc. the Fourth International Conference on Creating, Connecting and Collaborating through Computing (C5 2006),* January 2006, University of California, Berkeley, USA.
3. Xia, S., Sun, D., Sun, C. and Chen, D. 2005. “Collaborative object grouping in graphics editing systems,” In *Proc. The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, December 2005, San Jose, CA, USA.
4. Xia, S., Sun, D., Sun, C. and Chen, D. 2005. “Object-associated telepointer for real-time collaborative document editing systems”. In *Proc. The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, December 2005, San Jose, CA, USA.
5. Xia, S., Sun, D., Sun, C. and Chen, D. 2005. “Supporting workspace-mediated interaction in collaborative presentations with CoPowerPoint”. In *Proc. The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, December 2005, San Jose, CA, USA.
6. Xia, S., Sun, D., Sun, C. and Chen, D. 2005. “A collaborative table editing technique based on transparent adaptation”. In *Proc. The 13th International Conference on Cooperative Information Systems (CoopIS 2005),* LNCS, Springer Verlag, vol. 3760, pp. 576 – 592, November 2005, Agia Napa, Cyprus.

7. Xia, S., Sun, D., Sun, C., Chen, D. and Shen, H. 2004. “Leveraging single-user applications for multi-user collaboration: the CoWord approach”. In *Proc. ACM 2004 Conference on Computer Supported Cooperative Work*, pp. 162 – 171, November 2004 Chicago, IL USA.

8. Sun, D., Xia, S., Sun, C. and Chen, D. 2004. “Operational transformation for collaborative word processing”. In *Proc. ACM 2004 Conference on Computer Supported Cooperative Work*, November 2004, Chicago, IL USA.

9. Xia, S., Sun, D., Sun, C., Chen, D. and Shi, Y. 2004 “Interactive Presentation with CoPowerPoint”. *The Sixth International Workshop on Collaborative Editing Systems*, pp. 437 – 446, November 2004, Chicago, IL USA.

Abstract

People rely on off-the-shelf commercial single-user software systems in their daily lives and work to perform single-user tasks. People also need groupware systems to perform collaborative or group tasks. The goal of this thesis work is to develop innovative techniques for building computer applications that combine conventional single-user functionalities with advanced collaboration capabilities to effectively support people's individual and group work.

This thesis work contributes an innovative *Transparent Adaptation* (TA) approach and associated supporting techniques that can be used to convert existing or new single-user applications into real-time multi-user collaborative versions without changing their source code. The transparently adapted collaborative systems not only support unconstrained collaboration and other collaboration features that were previously seen only in advanced groupware research prototypes, but also maintain the conventional functionalities and interface features that were previously seen only in commercial off-the-shelf single-user applications. Major technical contributions of the TA approach include techniques for adapting the single-user application programming interface to the data and operation models of the underlying generic collaboration technique and a generic system architecture for collaborative systems.

The *Operation Transformation* (OT) technique has been chosen as the underlying collaboration technique for the TA approach due to its capability of supporting unconstrained collaboration and application independence. This thesis work has also made important contributions to OT by extending OT from supporting only collaborative plain text editing to supporting collaboration on complex data structures and comprehensive functionalities.

To support the adaptation of complex data and operation models in a range of applications, this thesis work has contributed a package of advanced adaptation techniques for collaborative table editing and graphic object grouping. These techniques have not only increased the capability of TA, but have also advanced the state-of-the-art of collaborative editing techniques.

To facilitate natural and smooth collaboration, this thesis work has contributed a multi-functional workspace awareness framework which is able to reduce the effort for developing workspace awareness features, and to be extended to support new workspace awareness features. Most importantly, this framework is able to deliver correct and precise workspace awareness information in the face of dynamic content and view changes in TA-based systems, which is an innovative feature unavailable in existing techniques.

The TA approach and supporting techniques were developed and tested in the process of transparently converting two commercial off-the-shelf single-user applications – Microsoft Word and PowerPoint – into real-time collaborative applications, called CoWord and CoPowerPoint, respectively. CoWord and CoPowerPoint not only retain the functionalities and the "look-and-feel" of their single-user counterparts, but also provide advanced multi-user collaboration capabilities for supporting multiple interaction paradigms, ranging from concurrent and free interaction to sequential and synchronized interaction, and for supporting detailed workspace awareness, including multi-user tele-pointers and radar views. The TA-based collaborative system architecture and the generic collaboration engine software component developed from this work can be reused in adapting a wide range of single-user applications.

Chapter 1

Introduction

Living in a social environment, people's everyday activities, including work, study and play, inevitably involve collaboration with others. As important tools for assisting people in complex tasks in the modern society, computer-based software systems also need to provide sufficient support to facilitate and enhance collaboration. These needs draw interests of social and computer scientists into a multi-disciplinary research field called *Computer-Supported Cooperative Work* (CSCW), which ranges from sociological analyses and anthropological descriptions on how people work in groups to the technological foundations of computer systems for supporting group work (Poltrock and Grudin 1994). These computer-based systems that support groups of people engaged in a common task (or goal) and that provide interfaces to shared environments are called groupware systems (Ellis et al. 1991).

This thesis explores technical issues about groupware systems that support geographically distributed users to work collaboratively in real time. In particular, it deals with the approach to leveraging single-user applications into real-time collaborative versions. The rest of this chapter describes the context of this research, the scope of this thesis and a summary of contributions.

1.1. Collaboration Awareness and Collaboration Transparency

Real-time groupware systems are built in two approaches: collaboration awareness and collaboration transparency (Lauwers and Lantz 1990). With the

collaboration-aware approach, groupware systems are developed especially for the purpose of supporting collaboration and collaboration mechanisms are *internal* to and *aware* by these systems and their designers. With the collaboration-transparent approach, on the other hand, groupware systems are based on existing (or new) single-user applications and collaboration mechanisms are *external* to and *unaware* by these applications and their designers.

Although it seems natural to develop special groupware systems to support collaboration, most existing collaboration-aware systems remain research prototypes, whose main purpose is to demonstrate novel collaboration techniques. Compared with off-the-shelf commercial single-user applications, functionalities of collaboration-aware systems for supporting conventional single-user activities (e.g. word processing, spreadsheet editing and graphics editing) are quite limited. Therefore, collaboration-transparent systems also play an important role in supporting collaboration. A significant advantage of collaboration-transparent systems is that they allow users to collaborate with their familiar single-user applications. This relieves users from the burden of learning new collaborative systems. As pointed out by Grudin (1994b), "who would abandon their favorite word processors to use a co-authorship application?" Moreover, by adding collaboration functionalities to existing single-user applications, the effort for developing groupware systems is significantly reduced.

1.2. Scope of This Thesis

1.2.1. Problem Statement

Existing collaboration-transparent approaches suffer from a series of problems, which prevent them from having comparative collaboration capabilities as collaboration-aware approaches.

Early collaboration-transparent systems, including NLS (Engelbart 1975), MMConf (Forsdick 1985; Crowley et al. 1990), Dialogo (Lantz 1986; Lauwers et

al. 1990), Share (Greenberg 1990), XTV (Abdel-Wahab and Peit 1991), Shared X (Garfinkel 1994), NetMeeting (Microsoft Corp. 2006a), SunForum (Sun Microsystems Inc. 2006a) and Timbuktu (Netopia Inc. 2006), provide *generic* application-sharing environments in which existing single-user applications can be transparently shared by multiple users for real-time collaborative work. The majority of these systems are designed to share existing single-user applications for supporting computer-based real-time conferences. Strict WYSIWIS (What You See Is What I See), modeled after the chalkboard in meetings, is a fundamental abstraction for multi-user interfaces and is supported by all these generic application-sharing systems. In a strict WYSIWIS mode, all users have to view exactly the same segment of the shared workspace. Moreover, to meet the needs of coordinated activities in many meeting processes, generic application-sharing systems support a *sequential interaction* paradigm, where only one user (i.e. the current holder of the *floor*) can interact with the shared application at any instant of time. Finally, some important collaboration functionalities that are supported in collaboration-aware systems, including detailed workspace awareness and flexible session management functions, are difficult to incorporate into generic application-sharing systems.

To solve these problems, Flexible JAMM (Begole et al. 1999) adopts a component replacement approach. The basic idea is to replace selected single-user components of the shared application with multi-user versions. These multi-user components are able to make use of application semantic knowledge to maintain consistency in the face of concurrent work and support relaxed WYSIWIS. With this approach, Flexible JAMM also supports some workspace awareness features (e.g. telepointer and radar view). Flexible JAMM requires that the underlying environment support process migration, run-time component replacement, dynamic binding, and interception/introduction of low-level user input events. However, single-user applications and execution platforms meeting Flexible JAMM requirements are limited. This approach cannot be applied to most commercial off-the-shelf single-user applications.

ICT (Li and Li 2002) is another framework for transparent sharing of existing single-user applications. In addition to the goal of achieving unconstrained collaboration and relaxed WYSIWIS view-sharing, ICT also attempts to address the heterogeneity and interoperability issues that arise from sharing different applications in the same session. In a heterogeneous collaboration environment, the strategy of applying the same sequence of input events at all collaborating sites for consistency maintenance, as used in homogeneous application-sharing systems, does not work any more. To address this heterogeneity problem, the ICT work proposed to devise a mechanism that is capable of "understanding" the semantic meaning of the user's inputs, so that the same user input semantics can be interpreted by different input event sequences at different applications. The *Operational Transformation* (OT) technique (Ellis and Gibbs 1989; Sun and Ellis 1998) was used in ICT to resolve consistency issues among concurrent editing operations. For the ICT approach to work, the meta knowledge for understanding the semantics of a specific application has to be formalized in advance. Due to the tremendous difficulties in knowledge formalization and other technical challenges, the ICT prototype preserves limited functionalities of the shared application: editing operations exchanged among different editors are limited to plain text insertion and deletion only.

1.2.2. Research Hypothesis

The research hypothesis of this thesis is that *transparently converted systems can not only have advanced collaboration capabilities that were previously seen only in collaboration-aware systems, but also maintain conventional functionalities and interface features that were previously seen only in commercial off-the-shelf single-user applications*.

The primary goal of this research is to develop techniques that can be used to convert existing or new single-user applications into multi-user collaborative systems which meet the following requirements:

(1) **Application compatibility**: the user interface features, functionalities, and document formats of the original single-user application should be retained.
(2) **Application transparency**: no change to the source code of the original single-user application is required.
(3) **Fast local response**: the response to the local user's interaction should be as fast as the original single-user application.
(4) **Unconstrained collaboration**: users should be allowed to perform any operations on any data objects at any time, which implies relaxed WYSIWIS and concurrent work.
(5) **Workspace awareness**: the system should support a variety of workspace awareness features so that the user knows who is in the workspace, where they are working, and what they are doing.
(6) **Session management**: the system should provide lightweight and flexible session management support with which users can manage collaboration sessions with little effort.
(7) **Flexible interaction control**: the system should provide a variety of interaction control paradigms/policies ranging from concurrent and free interaction to sequential and synchronized interaction in order to effectively facilitate different collaborative tasks.

1.2.3. Research Approach

In this research, an experiment-driven approach has been taken to examine the research hypothesis. Existing collaboration-transparent approaches and state-of-the-art collaboration techniques adopted in collaboration-aware systems were first studied. By comparing collaboration-transparent and -aware systems, the reasons that caused the performance and capability deficiencies in existing collaboration-transparent systems were then analyzed. Based on this analysis, an approach that is able to overcome these deficiencies was devised. Then a collaborative system based on this approach was designed and implemented. The system designed in this experiment was CoWord – a collaborative word processor converted from

Microsoft Word. CoWord achieved advanced collaboration features that were seen only in collaboration-aware systems, and at the same time preserved the user interface, document format and other functionalities of Microsoft Word. Based on the experiences from the first experiment, techniques invented and used in this experiment were summarized, refined and formulated into an innovative approach, named as *Transparent Adaptation* (TA).

To verify its generality, this TA approach was re-applied to another application, Microsoft PowerPoint, and CoPowerPoint – a collaborative slides authoring and presentation system – was developed. CoPowerPoint achieved the same effects as CoWord in a different set of functionalities, thus providing a testimony of the generality of the TA approach.

Then, these two experimental systems were used as the vehicles for studying and experimenting with new collaborative editing techniques. Requirements for new techniques were identified from the experimental systems. Afterwards, a collection of new collaboration techniques was designed, including collaborative table editing and graphics grouping techniques, as well as workspace awareness techniques. These techniques were implemented in these systems to test their correctness and feasibility. With the development of these new techniques, the experimental systems acquired richer collaboration functionalities than existing collaboration-aware systems.

These two experimental systems were publicly demonstrated on the Internet and freely distributed around the world. Usage feedback and data collected from users from all over the world provided valuable usability information and confirmed the usefulness of the TA approach and experimental systems.

1.3. Summary of Contribution

This dissertation makes the following major contributions:

(1) **The *Transparent Adaptation* (TA) approach**. This approach is able to transparently (i.e. without changing the source code) convert existing or new single-user applications into real-time collaborative systems which not only achieve effects previously only seen in collaboration-aware systems, but also preserve the conventional functionalities and interface features of single-user applications.

(2) **Extension to the *Operational Transformation* (OT) technique.** The OT technique is the cornerstone of the TA approach. This research has made several important extensions to the basic OT technique. These extensions leverage the OT technique from supporting collaborative plain text editing to supporting unconstrained collaboration on complex data structures and comprehensive editing operations.

(3) **Advanced adaptation techniques for complex application semantics.** The basic TA approach is able to adapt elementary editing functionalities (e.g. rich format text and simple graphics editing). To convert editing functionalities with complex semantics (e.g. table editing and graphic object grouping), a collection of advanced adaptation techniques have been designed in the TA framework. These techniques extend the capabilities of TA and the underlying OT, and also have contributions to the research of collaborative editing techniques.

(4) **A multi-functional workspace awareness framework for TA-based systems.** For supporting workspace awareness in TA-based systems, a multi-functional workspace awareness framework has been contributed in this thesis work. Compared with existing workspace awareness techniques, this framework has two unique features: (a) it is able to accommodate the dynamic content and view changes in TA-based systems to deliver accurate workspace awareness information; and (b) it can be easily extended to support a variety of existing and new workspace awareness features.

(5) **Two TA-based real-time collaborative systems: CoWord and CoPowerPoint.** As a result of the experiment-driven research, two TA-based real-time collaborative systems have been developed, which are CoWord

(CoWord Demo 2006) and CoPowerPoint (CoPowerPoint Demo 2006). CoWord is a collaborative word processor converted from Microsoft Word. CoPowerPoint is a collaborative slides authoring and presentation system converted from Microsoft PowerPoint. Furthermore, the generic collaboration engine shared by these two systems can be reused to provide generic collaboration support to other TA-based systems.

1.4. Dissertation Overview

This dissertation is organized as follows. First, representative prior researches are reviewed in Chapter 2 as the research background of this thesis work. The basic TA approach is discussed in Chapter 3. Next, extensions to the data and operation models of the basic OT technique for supporting TA are presented in Chapter 4. Issues and solutions for applying TA to complex application semantics, including collaborative table editing and collaborative graphic object grouping, are discussed in Chapter 5. In Chapter 6, a multi-functional workspace awareness framework for TA-based systems is discussed. The experimental systems developed in this research – CoWord and CoPowerPoint – are described in Chapter 7. Chapter 8 compares the TA approach with replicated generic application-sharing systems in dealing with a series of challenging problems, discusses the applicability of the TA approach to collaboration-transparent and collaboration-aware applications, and highlights its requirements and limitations. Finally, contributions and future work are summarized in Chapter 9.

Chapter 2

Research Background

This chapter reviews prior research relevant to this thesis work. This review serves as the research background of this thesis work.

2.1. CSCW and Groupware Overview

The idea of supporting collaboration with computer systems can be traced to the 1960s, when Douglas Engelbart illustrated the screen-sharing collaboration capability of the NLS demonstration (Engelbart and English 1968). The term *Computer-Supported Cooperative Work* (CSCW) appeared in the 1980s and soon became a multi-disciplinary research field. In addition to computer science researchers, it also attracted researchers from economy, social psychology, anthropology, organizational theory and education etc. (Grudin 1994a). In the following years, CSCW became a broad research field that ranges from sociological analysis on how people work in groups to computer-based technologies supporting people's group work. At the same time, the term *groupware* appeared to mean multi-user CSCW supporting software systems (Baecker 1992). Research on groupware is more specific. It focuses on technologies for designing and developing systems for supporting people's group work.

In recent decades, the fast development of computer hardware and software technologies, especially the explosive expansion of Internet, has boosted research on CSCW and groupware techniques. Numerous groupware systems have been developed as commercial products and research prototypes.

Based on the time and space natures of the collaboration they support, groupware systems can be classified as shown in Table 2.1.

Table 2.1 The time and space matrix of groupware systems (Ellis et al. 1991)

	Same Time	**Different Times**
Same Place	Face to Face Interactions	Asynchronous Interactions
Different Places	Synchronous Distributed Interaction	Asynchronous Distributed Interaction

In the time dimension, groupware systems can be classified as synchronous systems and asynchronous systems. Synchronous groupware systems, also known as real-time systems, support users collaborating at the same time. Example real-time systems include collaborative editors and instant messaging systems Asynchronous groupware systems, also known as non-real-time systems, support users working on the same task at different times. Example non-real-time systems include electronic mail and bulletin board systems.

In the space dimension, groupware systems can be classified as co-located and distributed systems. Co-located systems support users collaborating at the same place. One co-located system example is the meeting room-supporting system. Distributed systems allow users to collaborate from different places. One distributed system example is the tele-conference system that allows users to attend a computer-supported conference from geographically distributed sites.

This thesis work focuses on real-time distributed groupware systems (the shaded cell in Table 2.1). These systems are used to support geographically distributed users to collaboratively work on common tasks in real time. Real-time distributed groupware systems may be used to support collaboration in many application domains. Example systems include text editors (Leland et al. 1988; Sun et al. 1998), drawing systems (Greenberg et al. 1992; Chen and Sun 1999), multi-user domains (MUD) (Mehlenbacher et al. 1994), video conferencing

(Nguyen and Canny 2005), media spaces (Bly et al. 1993) and shared whiteboards (Elrod et al. 1992).

To meet the requirement of supporting group work, real-time distributed groupware systems have to handle complex issues that do not appear in single-user applications. Major design issues associated with the design and implementation of groupware systems will be discussed in the rest of this chapter.

2.2. Centralized and Replicated Architecture

Architectures of groupware systems fall into two categories – the centralized architecture and replicated architecture (Lauwers et al. 1990). With the centralized architecture (Figure 2.1), there is only one shared application instance, which is maintained at a central site. Other collaborating sites have only client end systems with limited functions. User input events to the shared application are forwarded to the central instance. Graphical output information is generated from the central shared application, then distributed to and displayed at the client end.

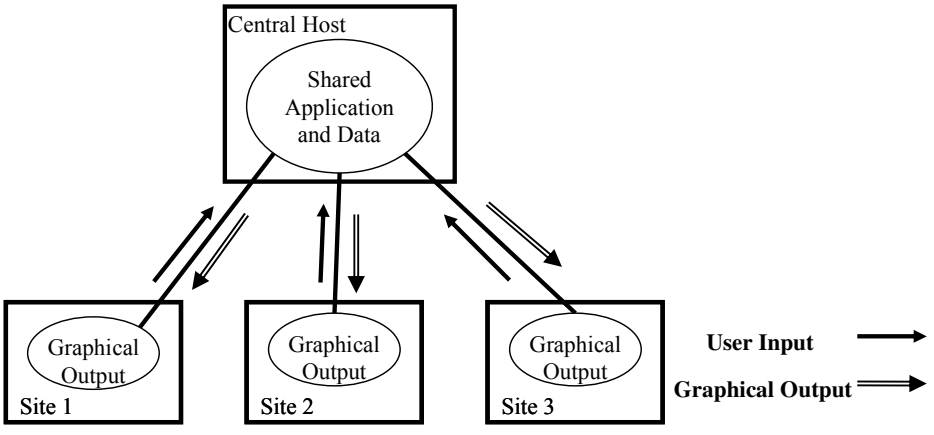


Figure 2.1 The centralized architectures.

The centralized architecture is easy to implement. Since there is only one instance of the shared application, developers need not worry about the system consistency. However, centralization also brings problems. The most significant disadvantage is the slow local responsiveness. Every local input event has to be sent to the central shared application; the local display cannot be updated until the graphical output from the central shared application is received. The local response time may be long with high network latency. Furthermore, centralized systems use network bandwidth inefficiently. This is because display information has to be broadcast to all collaborating sites, which usually consumes considerable network bandwidth (Lantz 1986; Begole et al. 1999). Finally, centralized systems may encounter compatibility problems if client sites have different hardware devices than the central site. Since the display information is generated at the central site, all client sites must have the capability to interpret this information, otherwise the graphical output may be displayed incorrectly.

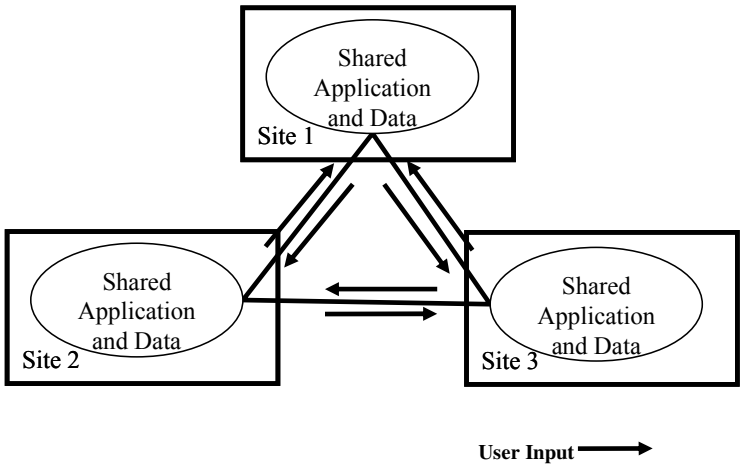


Figure 2.2 The replicated architecture.

These problems motivated the replicated architecture, in which each collaborating site has an instance of the shared application running at the local site, as shown in Figure 2.2. The replicated architecture is able to achieve fast local response because the user input events can be executed at the local site before being sent to remote sites. Application replication also provides the

possibility of supporting relaxed WYSIWIS, because each replica of the shared application may generate a different display. The use of network bandwidth is also improved because there is no need to broadcast display information.

However, the replicated architecture is not without its own problems. The major challenge is consistency maintenance. If users are allowed to interact with their local application replica freely, user input events may be executed in different orders among distributed sites. Maintaining consistency in the face of concurrent user input events is nontrivial. Moreover, when the replicated architecture is adopted in collaboration-transparent systems, more problems will occur (to be discussed in Section 2.6). In the following subsection, techniques for maintaining the system consistency will be reviewed.

2.3. Consistency Maintenance

Sun et al. (1996) have proposed a consistency model as a theoretical framework for consistency maintenance in replicated groupware systems. In this model, consistency is maintained by preserving the following properties.

(1) **Convergence**: When the same set of operations has been executed at all sites, all copies of the shared document are identical.
(2) **Causality preservation**: Operations are always executed in their natural causal order.
(3) **Intention preservation**: For any operation *O*, the effects of executing *O* at all sites are the same as the intention of *O*, and the effect of executing *O* does not change the effects of independent operations.

In essence, the *convergence* property ensures the consistency of the final results at the end of a collaboration session; the *causality preservation* property ensures the consistency of the execution orders of dependent operations during a collaboration session; and the *intention preservation* property ensures (1) that the effect of executing an operation at remote sites achieves the same effect as

executing this operation at the local site at the time of its generation, and (2) that the execution effects of independent operations do not interfere with each other. The consistency model imposes an execution order constraint on dependent operations only, but leaves it open for the execution order of independent operations as long as the convergence and intention preservation properties are maintained. The consistency model effectively specifies, on the one hand, what assurance a cooperative editing system promises to its users and, on the other hand, what properties the underlying consistency maintenance mechanisms must support (Sun et al. 1998).

There exist varieties of concurrency control techniques, which achieve some or all of the above consistency properties, as reviewed in the following subsections.

2.3.1. Floor Control

Floor control (Lauwers and Lantz 1990; Greenberg 1991), also called turn-taking (Greenberg and Marwood 1994), is a simple and coarse-grained concurrency control technique. With this technique, a user must acquire the token (the *floor*) before interacting with the shared workspace. Since there is only one floor in the system, only one user (the floor holder) can interact with the shared workspace at a time. Thus inconsistency problems are avoided. In addition to its simplicity, another advantage is the independency of applications, so it has been applied to a wide range of generic application-sharing systems, such as Microsoft NetMeeting and Hewlett-Packard Shared X (Garfinkel et al. 1994).

In systems adopting floor control, users have to interact with the system sequentially. Its applicability is limited to circumstances where a single active user is sufficient, such as computer-based conferences, but is not suitable for circumstances where high concurrency is required, such as collaborative document editing.

2.3.2. Locking

With the locking mechanism, a user must acquire a lock for an object before updating it. This technique avoids inconsistency problems by prohibiting concurrent accesses to the same object. Floor control can be regarded as an application-level locking in the sense that the active user acquires the lock for the whole system. Finer-grained locking applies locks on objects within the system such as text segments (Sun 2002b) or graphic objects (Chen and Sun 2001).

Locking mechanisms can be explicit or implicit. With explicit locking mechanisms, a user must explicitly acquire the lock before manipulating an object and explicitly release the lock after finishing manipulating the object. To relieve the user from the burden of these explicit actions, implicit lock mechanisms (Newman et al. 1992) allow the user to directly manipulate the object and the system implicitly acquires and releases the lock on behalf of the user.

Lock mechanisms can also be pessimistic and optimistic. Locking requests have to be sent to a central coordinator, which determines whether locking requests should be approved. With pessimistic locking, the user cannot manipulate the object until the locking permission is received from the coordinator, which may cause degradation of system responsiveness. Optimistic locking mechanisms (Greenberg and Marwood 1992; 1993) assume that the possibility of conflicts is very low and allow the user to manipulate the object before the locking request is approved. If conflicts occur, user actions are cancelled and the object state rolls back, which is an unpleasant experience for the user. Moreover, locking does not satisfy any of the consistency properties, because it only focuses on preventing conflicts within locked areas.

2.3.3. Serialization

With serialization mechanisms, operations generated by distributed users are executed in the same global order at all sites. In most serialization-based systems, the global execution order is derived from timestamps (Lamport 1978) of

operations. Some other systems determine the global execution order by recording the direct predecessor in every operation (Kanawati 1997).

Serialization can be pessimistic and optimistic. Pessimistic serialization mechanisms delay the execution of an operation until all its predecessors have been executed (Kanawati 1997). Since the execution of the local user's input may be delayed, this pessimistic strategy may lower the system's responsiveness. On the other hand, optimistic serialization mechanisms execute local user input immediately even if its predecessors remain unexecuted. To enforce the global order when an operation's predecessors are received after its execution, optimistic serialization mechanisms adopt an undo/do/redo strategy, which undoes the executed operation, executes its predecessors, and redoes the undone operation (Greenberg and Marwood 1993; Karsenty and Beaudouin-Lafon 1993). However, with this optimistic strategy, there is a possibility that the effect of an executed user input completely disappears in the undo/do/redo process.

Serialization can satisfy the convergence property, but cannot satisfy the causality and intention properties.

2.3.4. Operational Transformation

Operational Transformation (OT) (Ellis and Gibbs 1989; Sun and Ellis 1998) is an innovative optimistic concurrency control technique. With the OT technique, local operations are executed immediately after generation to achieve a high local responsiveness. Remote operations may need to be transformed against concurrent operations before execution, so that they achieve the same effects as in the local site. Combined with the causal ordering approach, OT is able to achieve all three consistency properties. In addition, OT supports undoing any operations (Sun 2002a).

The OT component in a collaborative editor is a complex system, but the basic idea of OT can be illustrated with a simple text editing scenario as follows. Given a text document with a string "abc" replicated at two collaborating sites; and two

concurrent operations: O_1 = *Insert*(0, "x") (to insert character "x" at position 0), and O_2 = *Delete*(3, "c") (to delete the character "c" at position 3) generated by two users at collaborating sites 1 and 2, respectively. Suppose the two operations are executed in the order of O_1 and O_2 (at site 1). After executing O_1, the document becomes "xabc". To execute O_2 after O_1, O_2 must be transformed against O_1 to become: O_2' = *Delete*(4, "c"), whose positional parameter is incremented by one due to the insertion of one character "x" by O_1. Executing O_2' on "xabc" will delete the correct character "c" and the document becomes "xab". However, if O_2 is executed without transformation, then it will incorrectly delete character "b", rather than "c". In summary, the basic idea of OT is to transform (or adjust) the parameters of an editing operation according to the effects of previously executed concurrent operations so that the transformed operation can achieve the correct effect and maintain document consistency.

An OT technique can be divided into two layers: the high-level transformation control algorithms, and the low-level transformation functions. Transformation control algorithms are responsible for determining which operation should be transformed against other operations according to their concurrency relationships; and transformation functions are responsible for determining how to transform one operation against another according to their operation types, parameters and other relationships. Varieties of transformation control algorithms have been presented in different OT techniques. Typical transformation control algorithms include dOPT (Ellis and Gibbs 1989), AdOPTed (Ressel et al. 1996), GOT (Sun et al. 1998), GOTO (Sun and Ellis 1998), SOCT2 (Suleiman et al. 1997), SOCT3/4 (Vidot et al. 2000), SDT (Li and Li 2004), LBT (Li and Li 2005a), ABT (Li and Li 2005b) and COT (Sun and Sun 2006). On the other hand, there are two types of transformation functions (Sun et al. 1998): one is the *Inclusive Transformation* function – IT (O_a, O_b), which transforms operation O_a against operation O_b in such a way that the impact of O_b is effectively included in the parameters of the output operation; and the other one is the *Exclusive Transformation* function – ET (O_a, O_b), which transforms operation O_a against

operation O_b in such a way that the impact of O_b is effectively excluded in the parameters of the output operation.

There are two underlying models in every OT technique: one is the *data model* that defines the way in which data objects in a document are addressed by operations; the other is the *operation model* that defines the set of operations that can be directly transformed by OT functions. Different OT techniques may have different data and operation models. For example, the OT techniques designed for supporting collaborative plain text editing (Ellis and Gibbs 1989; Ressel et al. 1996; Suleiman et al. 1998; Sun and Ellis 1998; Vidot et al. 2000; Ignat and Norrie 2003; Li and Li 2004) have an operation model consisting of two *Primitive Operations* (PO): *Insert* and *Delete*, and a data model of a single linear addressing space. Addresses in this linear addressing space ranges from 0 to $N-1$, where N is the total number of characters in the document. We use the term *basic OT technique* to mean these OT techniques defined for plain text editors. There are also OT techniques extended from the basic OT techniques used to support collaborative editing on more complex documents, such as spreadsheet (Fuller et al. 1993) and XML/HTML documents (Davis et al. 2002). These extended OT techniques have more complex data models and operation models.

2.4. Workspace Awareness

In a relaxed WYSIWIS view mode, users may work in different part of the shared workspace. In such circumstances, it is important that the user be aware of the status of other users so that they can collaborate naturally and fluently. Therefore, workspace awareness, which is defined as the up-to-the-moment understanding of another person's interaction with the shared workspace (Gutwin and Greenberg 1996; Gutwin et al. 1996a), plays an important role in groupware systems. Researchers have proved that workspace awareness significantly increases groupware usability (Gutwin and Greenberg 1999). Particularly, workspace awareness is used in groupware systems for managing coupling, simplifying communication, coordinating actions, helping users to anticipate future actions

and understanding assistances from others (Gutwin 1997; Gutwin and Greenberg 2002).

2.4.1. Workspace Awareness Information

According to Gutwin and Greenberg (2002), workspace awareness information consists of several items, as listed in Table 2.2.

Table 2.2. Composition of workspace awareness information (Grudin and Greenberg 2002).

Category	Element	Specific Questions
Who	Presence	Is anyone in the workspace?
	Identity	Who is participating? Who is that?
	Authorship	Who is doing that?
	Presence History	Who was here, and when?
What	Action	What are they doing?
	Intention	What goal is that action part of?
	Artifact	What object are they working on?
	Action History	How did that operation happen?
Where	Location	Where are they working?
	Gaze	Where are they looking?
	View	Where can they see?
	Reach	Where can they reach?
	Location History	Where has a person been?
How	Action History	How did that operation happen?
	Artifact History	How did this artifact come to be in this state?
When	Event History	When did that event happen?

The first column of Table 2.2 lists the basic categories of workspace awareness information, including *who* we are working with, *what* they are doing, *where* they are, *how* those events occur, and *when* various events happen. In each category, there are several specific workspace awareness knowledge elements, as shown in the second column. Each element can be described as the answer to a specific question about the shared workspace, as shown in the third column (Gutwin and Greenberg 2002).

2.4.2. Widely-Used Workspace Awareness Features

To deliver the above workspace awareness information, a wide range of workspace awareness features have been devised. Reviews of some frequently-used workspace awareness features follow.

(1) **Telepointer**. The telepointer is the avatar of a remote user's mouse cursor displayed on the local user's screens in real-time groupware systems. As an important groupware interface element, the telepointer is able to provide a range of group awareness information including *presence*, *location* and *activity*. In addition, telepointers can act as a communication channel by conveying gestural messages (Gutwin and Penner 2002; Gutwin et al. 2003). These features make telepointers a powerful means for providing users with a collaboration context and helping users coordinate the group work. Furthermore, researchers have made some extensions to improve the accuracy and expressiveness of the telepointer. Examples include (a) Smart Telepointer (Rodham and Olsen 1994), which associates the telepointer position with objects in the shared workspace to accommodate the view differences resulted from relaxed WYSIWIS, (b) Semantic Telepointer (Greenberg et al. 1996), which extends the representation form of the telepointer with changeable images or sound for delivering richer workspace awareness information, and (c) a series of techniques proposed by Dyck et al. (2004) and Gutwin et al. (2003) for improving the telepointer performance.

(2) **Multi-User Scrollbar**. The multi-user scrollbar (Baecker et al. 1993; Roseman and Greenberg 1996a) is an extension of the single-user scrollbar. It indicates remote users' view positions and sizes by displaying their scroll box positions in the local user's scroll shaft.

(3) **Radar View.** The radar view (Baecker et al. 1993; Roseman and Greenberg 1996a; Begole et al. 1999) is devised to deliver more detailed location information than the multi-user scrollbar in the two dimensional workspace. A radar view is often implemented as a miniature view of the shared workspace

with view ports of remote users. Each view port covers the view range in the shared workspace of a remote user. Furthermore, variations of the radar views (Gutwin et al. 1996b) were proposed to extend the expressiveness of the basic radar view.

In addition to the above widely-used workspace awareness features, researchers have proposed other ideas to deliver workspace awareness information. Examples include (1) *user list* (Ellis and Gibbs 1989; Isaacs et al. 1996; Roseman and Greenberg 1996b) which displays a list of all user information in the same collaboration session, (2) *sound* (Beaudouin-Lafon and Karsenty 1992), which is often used to deliver remote users' activity information, (3) *video embodiment* (Tang and Minneman 1990; 1991), which creates remote users' live images or shadows in the shared workspace, and (4) *Multi-user UI (User Interface) widgets* (Hill and Gutwin 2003), which are the multi-user versions of single-user UI widgets and deliver activity information of remote users in these widgets.

2.5. Session Management

Session management plays a key role in groupware systems. It determines how collaboration sessions are initiated and terminated and how individuals join and leave a session (Patterson et al. 1990). Existing session management approaches fall into two categories (Edwards 1994) – explicit and implicit session management approaches.

2.5.1. Explicit Session Management

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 (Crowley et al. 1990) adopts an *initiator-based* approach. After the initiating user creates a session, he/she sends invitations to other users, who will choose to accept or reject the invitations. The session begins after all invited users have responded or time out. On the other hand, Collage (NCSA 2005) 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.

Explicit session management approaches are suitable for supporting formal collaboration because they facilitate explicit session-related actions. However, they are not suitable for the spontaneous and impromptu collaboration circumstances due to their lack of flexibility. Furthermore, explicit inviting or joining actions involve too much overhead, which decreases the system usability. Finally, these approaches provide hardly any session awareness information.

2.5.2. Implicit Session Management

Implicit session management approaches are designed to avoid the overhead of explicit session management approaches and to support spontaneous and impromptu collaboration. These approaches manage collaboration sessions based on users' actions in the collaborative environment. According to the types of user actions they utilize to manage sessions, 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 typical example is Intermezzo (Edwards 1994). 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 into the same session. These approaches are frequently adopted in virtual space systems such as MASSIVE (Greenhalgh and Benford 1995). In MASSIVE, users are represented as objects equipped with communication media in a virtual 3D world. When objects are approximate enough and they support common communication media, a peer connection (i.e. collaboration session) is established between them.

Rusken (Texier and Plouzeau 2003) adopts a hybrid of artifact-based and place-based approaches. In the artifact-based aspect, Rusken extends the concept of *object* in implicit session management to *objects set*. Users accessing the same object set are joined into the same session. In the 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 (Isaacs et al. 1996). 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.

The major problem of implicit session management approaches is that they cannot provide users with sufficient session awareness information. Without the session awareness information, it is the system rather than users who determines whether to collaborate (Gutwin et al. 2005). 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 accessing an object.

2.6. Collaboration Transparency

Collaboration-transparent approaches aim to share existing single-user applications among distributed users. In the past decades, collaboration-transparent techniques have been developed in several generations.

2.6.1. Centralized Generic Application Sharing

A wide range of early collaboration systems provide generic application-sharing environments in which any existing single-user application can be transparently shared by multiple users in real-time collaborative work. Most of these systems adopt the centralized architecture (e.g. Microsoft NetMeeting, Sun Forum, Real VNC (RealVNC Ltd. 2006), HP Shared X (Garfinkel et al. 1994), Rendezvous (Patterson et al. 1990), Share (Greenberg 1990) and XTV (Abdel-Wahab and Peit. 1991)).

The technique used to transmit and display graphical output from the central shared application is called *display broadcasting* (Begole et al. 1999). In X Window-based generic application-sharing systems, display broadcasting is implemented by taking advantages of the separation of the X Client – the application process, and the X Server – the process that handles graphical requests from X Clients and generates display output. On the other hand, in MS Windows-based generic application-sharing systems, display broadcasting is usually implemented based on ITU T.128 or its extended versions, which are conceptually similar to its counterpart in the X-Window.

In addition to the problems resulted from the centralized architecture (see Section 2.2), workspace awareness features supported in these systems are limited to the telepointer (Crowley et al. 1990). In the strict WYSIWIS view-sharing mode, all users are viewing the same segment of the shared workspace. There is no need to provide other workspace awareness features to indicate view positions and ranges of different users.

2.6.2. Replicated Generic Application Sharing

In attempts to deal with problems resulted from the centralized architecture, some later systems adopt the replicated architecture in which each collaborating site has an instance of the shared application. Examples include Dialogo (Lauwers et al. 1990), MMConf (Crowley et al. 1990), VConf (Lantz 1986) and Rapport (Ahuja

et al. 1990). In contrast to the *display broadcasting* technique used in centralized systems, these systems adopt the *event broadcasting* technique, where input events to the local site are broadcast to all remote sites.

In addition to the difficulty in consistency maintenance in the face of user inputs, replicated generic application-sharing systems also encountered challenges in maintaining consistency in the face of inputs from non-user external resources, such as files, clocks, environment variables or network connections. Running in different execution environments, replicas of the shared application may receive different inputs from external resources, which breaches the system consistency. This problem is known as *externalities (*Begole et al. 2001). Furthermore, when a newcomer is to join an ongoing collaboration session, it needs the same execution environment as existing sites. This problem, known as *late-comer*, is also nontrivial in collaboration-transparent systems.

These problems were once regarded as intractable in early generic application-sharing systems (Lauwers et al. 1990). Researchers have presented varieties of solutions.

To maintain the system consistency in the face of user inputs, replicated generic application-sharing systems have to ensure that all replicas receive user inputs in the same order. To achieve this goal, explicit or implicit floor control mechanisms are adopted. However, with floor control, only the user who has the floor can interact with the system, which results in the *sequential interaction* problem and the inability to support concurrent work (Begole et al. 1999, Sun et al. 2006; Xia et al. 2004).

To handle the externalities problem, replicated generic application-sharing systems create identical execution environment for all participants automatically (Crowley et al. 1990; Lauwers et al. 1990) or manually (Lantz 1986). However, these solutions are ad-hoc and they only handle the file accessing externalities problem.

To accommodate late-comers, replicated generic application-sharing systems usually adopt two late-joiner-accommodating techniques, including (1) *event replay* (Chung et al. 1993), which records all input events to an existing site and replays these events to the joining site, and (2) *image copy* (Chung and Dewan 1996), which copies the process image in the memory of an existing site and imports it to the joining site. However, these approaches suffer from different problems. Event replay approaches are inefficient. The performance could degrade drastically in the face of long execution time and potentially expensive operations (Begole et al. 1999). Image copy approaches are not widely applicable. They require support from special underlying execution environments (Douglis 1990; Milojičić et al. 1993) or from development tools (Bharat and Cardelli 1995; Zhang and Pande 2005).

2.6.3. Component Replacement

Flexible JAMM (Begole et al. 1999) represents a shift from seeking generic solutions at the operating/windowing system level to exploring solutions at the application interface library level. It adopts a replicated architecture for achieving fast local response and efficient network usage.

The major innovation of Flexible JAMM is the *component replacement* approach, which replaces selected user interface components (e.g. buttons or text panes) of the shared application with collaboration-aware ones at runtime. These collaboration-aware components understand the application semantics so that they are able to solve most of the replication-related problems. First, these collaboration-aware components can selectively broadcast user input events to other collaborating sites. In addition to reducing the consumption of network bandwidth, this event-filtering technique is also able to achieve relaxed WYSIWIS by filtering events that do not affect the system consistency (e.g. scrolling events). Second, a collaboration-aware text editing pane embedded with the Operational Transformation technique is able to support unconstrained real-time collaborative text editing. Third, based on the object migration capability of

Swing and JOS (Java Object Serialization), late comers can be accommodated by migrating application objects from an existing site. Finally, this approach is able to support detailed workspace awareness features, including the telepointer and radar view.

Flexible JAMM also applies the object replacement combined with a proxy approach to handle the externalities problem (Begole et al. 2001). An external resource is wrapped in an externality server. External resource accessing objects in the shared application are replaced with externality proxies, which always acquire data from the externality server. Both externality proxies and servers are collaboration-aware, so that the inconsistency possibility related to external resources can be avoided.

The object replacement approach has its own limitations: (1) objects created after sharing cannot be replaced; and (2) subclasses of replaceable classes cannot be replaced. Moreover, this approach requires special supports from the execution platform. Therefore, this approach is not widely applicable. Currently, it can only be applied to Swing-based Java applications.

2.6.4. Collaboration Transparency and Heterogeneity

The above collaboration-transparent systems are homogeneous ones since they require users to collaborate with the same shared application. While users are allowed to share different single-user applications in the same session, the heterogeneous issue arises.

In addition to its goal of achieving unconstrained collaboration and relaxed WYSIWIS view-sharing, ICT (Li and Li 2002) attempts to address the heterogeneity and interoperability issues in collaboration-transparent systems. The main challenge is that the event interception and replay approach used in generic application-sharing systems no longer works in heterogeneous environments because different applications process events in different ways. The

solution adopted in ICT is to devise a mechanism that is able to understand the semantics of the user input events. With this mechanism, the heterogeneity issue is addressed by translating local system-specific events into higher-level operations at the local site, and translating operations into remote system-specific events at the remote site. Furthermore, the consistency maintenance issue is addressed by processing operations with the Operational Transformation technique.

However, discovering and formalizing semantic knowledge of commercial off-the-shelf single-user application's functionalities and interface features are tremendously difficult while the shared application is assumed as a black box. Due to this problem, ICT can only preserve limited conventional functionalities and user interface features of the shared application. The ICT prototype, which supports interoperation between MS Word and GVim, is limited to supporting collaborative plain text insertion and deletion only.

2.7. Summary

This chapter has reviewed relevant prior research on CSCW and groupware techniques.

CSCW is a broad research field that ranges from sociological analysis to computer-based technologies, while groupware research focuses on technologies for designing and implementing systems for supporting people's group work. From the time dimension, groupware systems can be classified as real-time and non-real-time systems. From the space dimension, they can be classified as co-located and distributed systems.

Groupware systems adopt two architectures. With the centralized architecture, there is only one instance of the shared application maintained at a central site. With the distributed architecture, each collaborating site has an instance of the shared application. The centralized architecture has a series of problems,

including slow local response, inefficient network bandwidth use and compatibility. On the other hand, the major problem of replication is the difficulty of consistency maintenance.

System consistency can be described with a consistency model with three properties: convergence, causality preservation and intension preservation. Major consistency maintenance mechanisms include (1) floor control, in which a user must obtain the token (the *floor*) before interacting with the shared workspace, (2) locking, in which a user must acquire a lock for an object before manipulating it, (3) serialization, which forces operations generated by distributed sites to be executed in the same global order at all sites, and (4) Operational Transformation, which adjusts parameters of editing operations according to previous executed concurrent operations.

While WYSIWIS is relaxed, workspace awareness is important to improve groupware systems' usability. Different workspace awareness features are able to deliver presence, location and activity information of others in the same collaboration session. Widely-used workspace awareness features include telepointer, multi-user scrollbar and radar view.

Session management determines how collaboration sessions are initiated and terminated and how individuals join and leave a session. Explicit session management requires users to take explicit session-related actions, so it cannot support spontaneous and impromptu collaboration. Implicit session management solves these problems by implicitly managing collaboration sessions according to users' object accessing actions, but it cannot provide sufficient session awareness information.

Collaboration transparency is an approach to developing groupware systems by converting existing single-user applications into collaborative versions without changing their source code. Most of collaboration-transparent systems are generic application-sharing systems. Centralized generic application-sharing systems

adopt the display broadcasting technique to deliver the graphics output from the central instance to collaborating sites. Replicated generic application-sharing systems adopt the event broadcasting technique to deliver user input from the local site to other collaborating sites.

Generic application-sharing systems with the replicated architecture have difficulties in handling several problems, including consistency maintenance, accommodating late-comers and externalities. Flexible JAMM handles these problems by dynamically replacing selected components of the single-user application at runtime. However, the major problem of this component-replacing approach is its special requirements to the execution environment, so it is not widely applicable. ICT attempts to address the heterogeneous issues arising when different single-user applications are shared in the same session with a mechanism that understands the semantics of the shared application. Due to the tremendous difficulty in knowledge discovering and formalizing, the ICT prototype only preserves limited functionalities of the shared application.

Chapter 3

The Transparent Adaptation Approach

The *Transparent Adaptation* (TA) approach is the central contribution of this thesis work. It was designed to leverage existing or new single-user applications for multi-user real-time collaboration. Based on this approach, two collaborative editing systems, CoWord, which is a collaborative word processor, and CoPowerPoint, which is a collaborative slides authoring and presentation system, have been developed. This chapter takes these two systems as examples to discuss the TA approach.

3.1. Introduction

Unlike existing application-sharing approaches, the TA approach tackles the transparent conversion of single-user applications from a different angle. Rather than endeavoring to share any single-user application in an operating/windowing system (e.g. NetMeeting and SunForum) or with a library (e.g. Flexible JAMM), the TA approach transparently converts individual single-user applications into collaborative versions. The major benefit of the relaxation of the generic application-sharing constraint is the possibility of taking advantage of application semantic knowledge and introducing application-specific treatment to the target application, so that some challenging problems associated with the generic application-sharing environments are significantly simplified or completely avoided. Moreover, to postpone dealing with complex collaboration issues in

heterogeneous environments (Knister and Prakash 1990; Li and Li 2002), the current TA work is restricted to homogeneous collaboration environments in which all users are required to use the same converted application in a collaboration session.

The TA approach is based on (1) the use of the single-user application's API (Application Programming Interface) to intercept and replay the user's operations, so it requires no access or change to the application's source code (thus being transparent), and (2) the use of *Operational Transformation* (OT) to manipulate intercepted user operations for supporting responsive and unconstrained (i.e. concurrent and free) multi-user interactions with the shared application. For the TA approach to work, however, the shared application's API must be adaptable to the data model and operation model of the OT technique. With the support of OT, TA-based collaborative applications are able to achieve fast local response, concurrent work, relaxed WYSIWIS, and detailed workspace awareness.

Microsoft Word was chosen as the first target single-user application for transparent adaptation. This is because word processors are among the most commonly used single-user applications, and Word provides a set of comprehensive, complex, and interesting data types, operations, and a sophisticated API for investigation. Our goal is to covert Word into a real-time collaborative word processor, called CoWord, which allows multiple users to view and edit any objects in the same Word document at any time over the Internet. As a follow-up of CoWord, the TA approach was re-applied to convert Microsoft PowerPoint into CoPowerPoint – a multi-user real-time slides authoring and presentation system. The CoPowerPoint work tested the generality of the TA approach and provided new insights and solutions for adapting different classes of applications.

The rest of this chapter is organized as follows. First, data model adaptation issues and techniques in CoWord and CoPowerPoint are discussed in Section 3.2. Then, operation model adaptation issues and techniques in CoWord and

CoPowerPoint are discussed in Section 3.3. Finally, this chapter concludes with a summary in Section 3.4.

3.2. The Data Model Adaptation

As one of the major technical components of the TA approach, the data model adaptation is responsible for bridging the gap between the API addressing schemes and the OT data model. In this section, some basic ideas and techniques learnt from adapting two different applications, Word and PowerPoint, will be discussed.

3.2.1. Word Data Model Adaptation

A Word Document from the User's View

Unlike a plain text document, where all characters are presented at the user interface in a linear sequence, a Word document, when viewed by a user, does not always look like a linear sequence of objects. For example, graphic objects may appear at any position in the document's two-dimensional display space. Furthermore, a graphic object may be moved freely from one location to another without affecting the locations of other objects, which is different from moving (inserting and deleting) a character in a string. As shown in Figure 3.1, the user's view of a Word document consists of some sequences of formatted character objects (e.g. "CoWord, a collaborative word processor"), some graphic objects that are inline with character sequences (e.g. the "Welcome" ClipArt object that is inline with the sequence of characters "To CoWord"), and some graphic objects that are floating in the two-dimensional space and may overlap with each other (e.g. the Textbox with text "Word" that is on top of another ClipArt object).

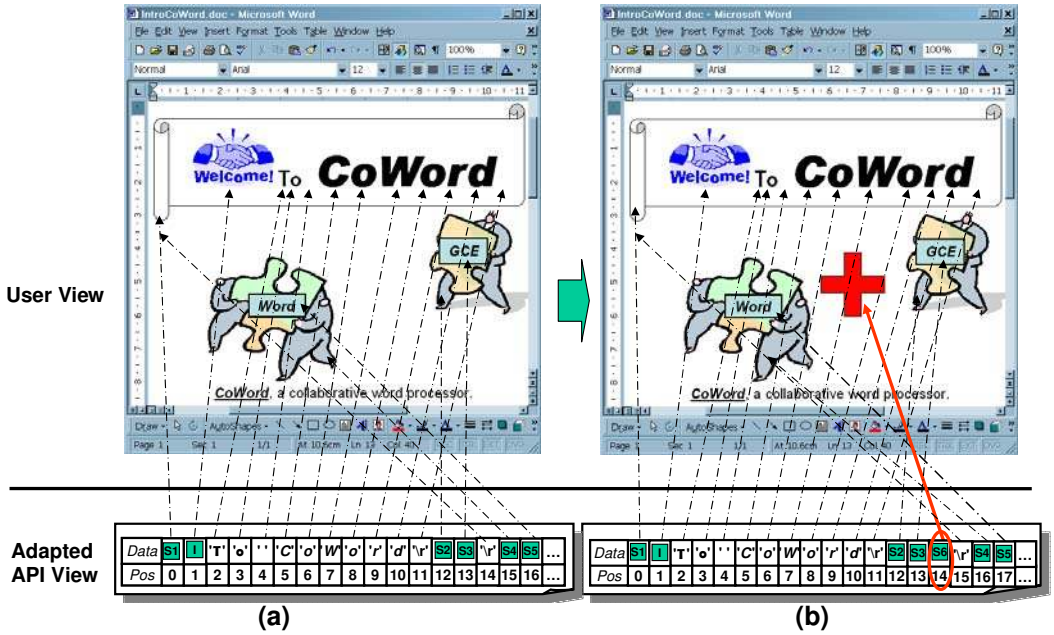


Figure 3.1 The user's view and the adapted API's view of a Word document.

This irregular and arbitrary presentation/view of data objects in the Word document appears to be a major obstacle for applying OT to Word documents since this view does not match the linear addressing space of the basic OT data model. However, our investigation discovered that the presentation of data objects at the user interface is actually irrelevant to the applicability of OT. What really matters is how data objects are addressed from the application's API.

A Word Document from the API's View

Word provides a comprehensive API which conforms with Component Object Model (COM) Automation (Iseminger 2000). With this API, software developers can change the behavior of the application, enhance the application's functionality, or incorporate the application into other applications. In particular, this API provides high level interfaces for accessing and manipulating data objects in a Word document.

From the Word API's view, data objects of various types (e.g. text, ClipArt objects, Drawing objects, and WordArt objects) are modeled by some basic

objects (Microsoft Corp. 2006b), including *Text*[1] (e.g. a sequence of formatted characters), *InlineShape* (e.g. a ClipArt object embedded in a sequence of characters), and *Shape* (e.g. a floating graphic object). For the purpose of address adaptation, the most relevant feature of this API is the ability to access all data objects from a global linear addressing space by means of a *Range* object. All *Text* objects and *InlineShape* objects are displayed sequentially in the document, and can be accessed by their position references in the *Range*-based linear addressing space; floating graphic objects (i.e. *Shape* objects) are displayed at arbitrary positions in the drawing layer of the document, but they have corresponding anchors in the *Range*-based linear addressing space. From these anchors, the corresponding floating objects can be accessed.

The relationship between the data objects at the user interface and their position references in the *Range*-based linear addressing space from the Word API is illustrated in Figure 3.1. Every data object at the Word user interface has a corresponding position reference in the linear addressing space of the Word API. For example, the floating ClipArt object at the left-top location of the drawing space has an anchor, denoted by *S1*, at position "0" of the linear addressing space; the "Welcome" inline ClipArt object has a position reference "1"; the inline character "T" has a position reference "2", and so on.

When the user draws a new floating graphic object (the "+" sign) in the drawing layer of the document (the user view in Figure 3.1-(b)), this object's anchor, denoted by *S6*, is automatically inserted at a suitable position ("14" in this example) in the linear addressing space. Meanwhile, other objects' position references on or higher than the new anchor's position are shifted to the right by one position, as shown in the Word API's *Range*-based linear addressing space in Figure 3.1-(b). If an object is removed from the document, its position in the linear addressing space will be removed and all other objects' position references

[1] In fact, *text* is treated as part of the *Range* object, rather than as a separate object in the Word API. *Text is treated* as an object for the sake of convenience.

on the right of the removed object will be shifted to the left by one position (not shown in Figure 3.1).

It is worth pointing out that the Word API also provides alternative ways to access floating objects (e.g. by their unique names). However, creating/removing a floating object in/from the drawing space always results in inserting/deleting an anchor to/from the global linear addressing space, which unavoidably has an impact on the positions of other objects in the document. Therefore, the anchor's position must be used as the identifier of editing operations for floating objects in order to use OT for concurrency control of all editing operations in CoWord. In other words, the use of the position references in the *Range*-based linear addressing space is not only sufficient but also necessary to address all types of data objects in CoWord.

A Tree of Linear Addressing Domains for a Word Document

Apart from the main body of a Word document, there are also other auxiliary document elements, such as *Comments*, *Footnotes*, *Headers*, and *Footers*, which are displayed in designated locations of the document. The user can annotate the document by attaching *Comments* or *Footnotes* to selected text segments, or break the document into multiple *Sections* and associate different *Headers* and *Footers* with these sections, etc. At the user interface, these elements are interrelated and form integral parts of the document.

For the purpose of address adaptation, these elements can be viewed as mutually independent editing areas: operations performed on data objects in one element have no impact on the data objects in other elements. After similar address adaptation analysis was applied to these elements, it was found that data objects in each element form a linear addressing domain as well. Moreover, it was found that these elements are linked to the main body of the document by special links, each of which occupies one position in the main body document.

Consequently, the main body of the Word document and all auxiliary document elements form a tree of linear addressing domains, as shown in Figure 3.2. The top layer of the tree contains the linear addressing domain corresponding to the main body of the document. The second layer of the tree consists of multiple independent linear addressing domains corresponding to *Comments*, *Footnotes*, *Headers*, and *Footers*, etc. For each second layer domain, there is a corresponding link in the top layer domain. For example, a link for a *Comment* or a *Footnote* occupies one position in the top linear addressing domain, and provides a reference to the *Comment* or *Footnote* itself. A *Section-Break* link also occupies one position in the top linear addressing domain, and provides a reference to a collection of *Headers* and *Footers* associated with the corresponding section. Each of the *Headers* and *Footers* can be identified by its unique name (determined by the Word API), and forms one independent linear addressing domain.

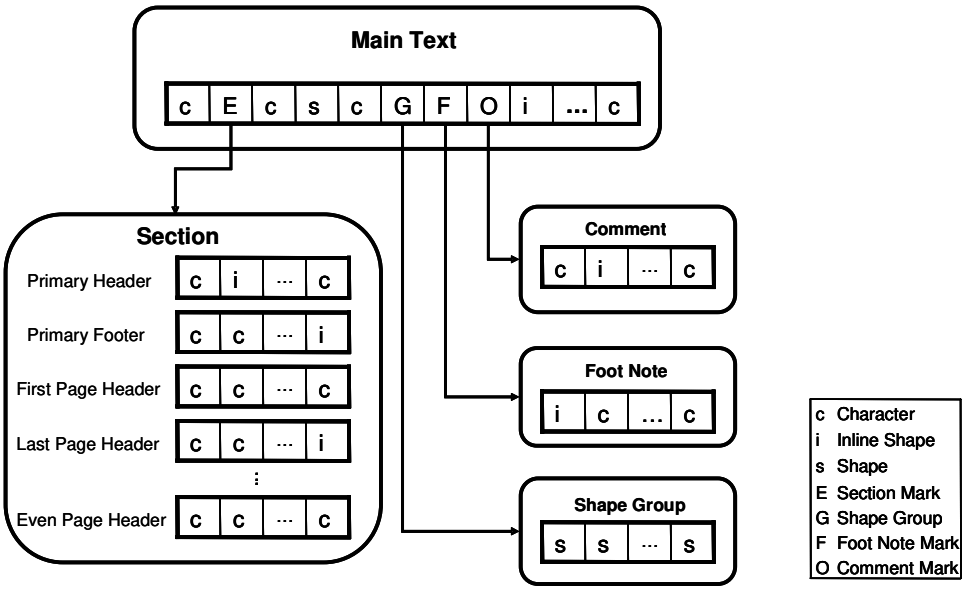


Figure 3.2 A tree of linear addressing domains for a Word document.

Based on the data model in Figure 3.2, to access a data object in the main body of the document, the position reference of this data object in the top linear addressing domain is needed. To access a data object in a *Comment* or *Footnote*,

two position references are needed: one is the position of the *Comment* or *Footnote* link in the top linear addressing domain, and the other is the position of the data object itself inside the addressing domain corresponding to the *Comment* or *Footnote*. To access a data object in a *Header* of a section, the following pieces of information are needed: the position reference of the corresponding *Section-Break* link in the top linear addressing domain, the unique *Name* of the *Header*, and the position of the data object itself inside the linear addressing domain corresponding to the *Header*.

3.2.2. PowerPoint Data Model Adaptation

PowerPoint is different from Word in its functionalities, user interfaces, and API, thus providing a vehicle for investigating and illustrating the diversity of the data address adaptation techniques in different applications.

PowerPoint User Interface and API

PowerPoint provides the user with multiple levels of interfaces, called views, to edit or show the document. One editing interface is the *slide-sorter-view*, as shown in Figure 3.3-(a)-(1). In this view, a PowerPoint document is presented as a sequence of slides. The granularity of the user's actions in this view is at the slide level. For example, the user can insert or delete slides, re-arrange the order of slides, or customize the design template or background of all slides.

From the *slide-sorter-view* interface, the user can "zoom" into any individual slide to edit the graphic objects in that particular slide. Another view, called *normal-view*, is provided for users to edit graphic objects inside a slide, as shown in Figure 3.3-(a)-(2). From this view, the user can create, remove, or change any graphic objects in a slide, including Textboxes, ClipArts, etc. In addition to the drawing space, each slide is also associated with a separate *Notes* area for the user to write explanatory notes for the corresponding slide.

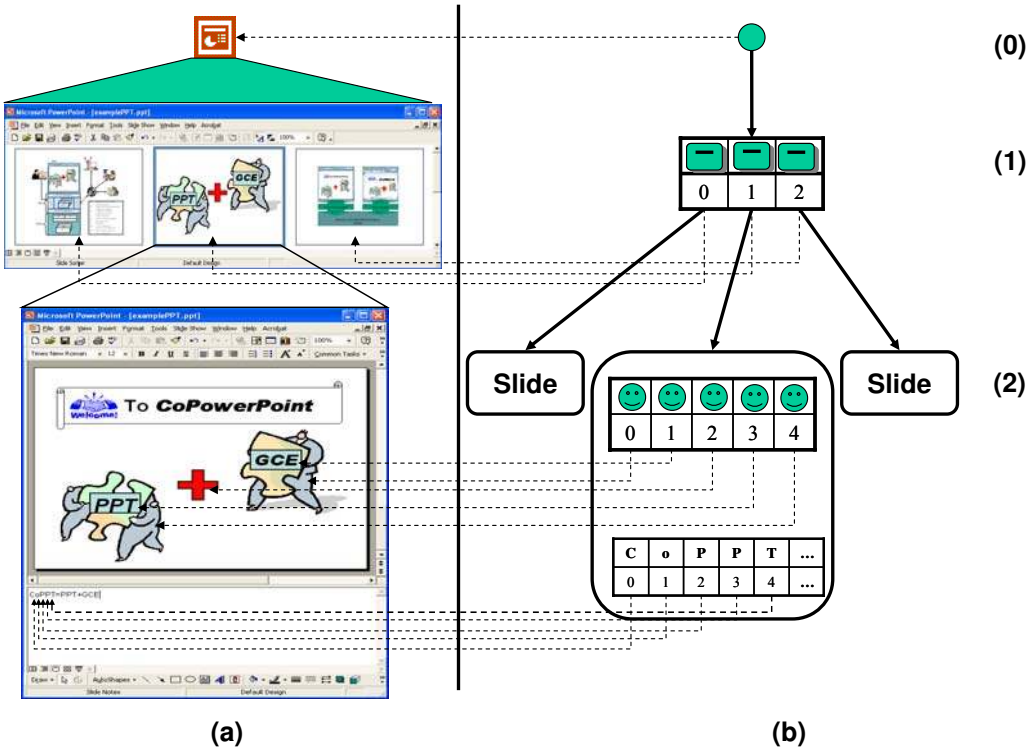


Figure 3.3 (a) The user's views of a PowerPoint document. (b) The API's view of a PowerPoint document.

Apart from these editing views, there is another presentation interface, called *slide-show*. From this *slide-show* interface, the user can control the presentation (e.g. go to the next, previous, or a specific slide, animation, or annotate the presentation screen with a virtual pen), but cannot change the contents of slides.

Despite the various differences in these user interfaces, the data objects being viewed from different views belong to the same document and are accessible in the same way from the PowerPoint API. For example, the same graphic object in a slide can be viewed by the user from the *slide-sorter-view*, *normal-view*, or *slide-show*. However, there is only one internal representation of this graphic object in the document and it can be addressed from the API in the same way, regardless from which view it is accessed by the user. From the PowerPoint API, a three-level hierarchical structure of the data objects in the PowerPoint document can be extracted: slide sequence, individual slides, and individual graphic objects. The following address adaptation discussions will be organized according to this

three-level hierarchical structure of the data objects in PowerPoint, rather than the different views at the user interface.

Addressing Slides in the Slides Sequence

From both the user interface and the API, the slides in a PowerPoint document are organized as a sequence, shown in Figure 3.3-(1), which directly matches the basic OT data model. Apart from the sequence of normal slides, there are some special *master* slides at the top level of a PowerPoint document, including *Slide Master*, *Title Master*, *Handout Master*, and *Notes Master*. The contents of these *master* slides are integrated with normal slides in the user interface presentation, but, from the API's view, these top-level *masters* are independent of the normal slides and independent of each other. Data objects in these *masters* can be edited and addressed in similar ways as in other normal sides, as discussed in the next subsection.

Addressing Graphic Objects inside Individual Slides

At the individual slide level, there are two independent editing areas: one is the graphic object drawing area, and the other is the explanatory *Notes* area. The *Notes* area is a text editing area, as shown in Figure 3.3-(2), which directly matches the basic OT data model. The following discussion focuses on addressing graphic objects in the drawing area.

Unlike the slides sequence or the text editing area, graphic objects in a slide drawing area do not appear to be organized in any sequence at the user interface. Similar to the *Range*-based addressing scheme for floating objects in the Word API (see Section 3.2.1), the PowerPoint API also supports an index-addressing scheme, which can be used to address graphic objects in a slide sequentially. For example, the slide in Figure 3.3-(a)-(2) contains five graphic objects, which can be addressed by index-addresses: 0, 1, 2, 3 and 4, as shown in Figure 3.3-(b)-(2). An important property of this scheme is that index-addresses are interrelated like the positions of characters in a string. The creation of a new graphic object or the

removal of an existing graphic object may change the index-addresses of those objects with index-addresses larger than the created/deleted object. The change of an existing object's attribute (e.g. color, size, font) will have no effect on the index-addressing space. Clearly, the index-addressing space matches very well with the basic OT data model. This is another example where the data objects may be presented at the user interface in a non-sequential way but can be addressed sequentially from the API.

It is worth pointing out that the PowerPoint API also provides another name-addressing scheme: every existing graphic object can be addressed by its unique name, which is assigned at the time of creating this object. An important property of the name-addressing scheme is that names are independent, which means that creation of a new graphic object or deletion of an existing one from a slide does not affect the names of other objects. If the independent name-addressing scheme were used to access graphic objects in a slide, then there would be no need for using OT to ensure consistency at this level (Sun and Chen 2002). A question arises: why not use this name-addressing scheme to access graphic objects in CoPowerPoint? The main reason against using the name-addressing scheme is that this scheme is incapable of addressing multiple replicas of the same object at different sites. This is because replicas of the same group of data objects may be created in different orders in an unconstrained collaboration session, and these replicas may be assigned different local names by their respective local PowerPoint. To support collaborative editing of replicated objects based on the name-addressing scheme, an additional global object naming scheme for all replicated objects and corresponding consistency maintenance techniques has to be devised, which is nontrivial. The index-addressing scheme is preferred because it allows the use of the same established OT technique at all levels, thus saving the trouble of having to devise and test new techniques as required by the name-addressing scheme.

Another reason for choosing the index-addressing scheme is the need to ensure consistent z-order-values of replicated graphic objects inside a slide. The z-order

values of objects represent their relative layering in the z-dimension of the drawing space; the z-order-values range continuously from 0 to $N-1$, where N is the total number of objects in a slide. When a new graphic object is created, it is initially assigned the current largest z-order-value and placed at the top of the z-dimension of the drawing space. In an unconstrained collaboration environment, if no special measure is taken, the z-order values of objects (i.e. their overlapping relationships) may become inconsistent at different sites. For example, consider two graphic objects G_1 and G_2 created concurrently by two users. Suppose these two objects are overlapping. After the two objects are created at both sites in different orders, G_1 will be on top of G_2 at the site where G_1 was created last; and G_2 will be on top of G_1 at the site where G_2 was created last. Moreover, z-order inconsistency may also occur when users concurrently change the z-order-values of existing objects (e.g. by invoking "Bring to Front" or "Send to Back" interface commands). The z-order inconsistency problem is the same in nature as the inconsistency problem encountered in performing concurrent insertion and deletion operations in any sequence. Therefore, OT is needed here to ensure consistent z-order values of replicated objects.

In the PowerPoint API, the index-address of a graphic object has the same value as its z-order-value. Therefore, the index-addressing scheme combined with the OT technique can not only correctly identify replicated objects, but can also consistently maintain the z-order values for all graphic objects in PowerPoint.

Addressing Internal Structures of Individual Graphic Objects

Individual graphic objects may have internal structures that can be manipulated by PowerPoint built-in operations or by external applications. For example, a *Textbox* object contains a sequence of formatted characters, to which various built-in editing operations can be applied. Clearly, the sequence of characters in a *Textbox* forms a linear addressing domain at a lower layer, to which the basic OT technique can be applied in order to merge concurrent operations at this layer.

However, not all graphic objects can be treated in this way. If the internal structure of certain graphic objects is inaccessible from the API (e.g. objects created by external applications), or cannot be modeled as a linear addressing domain (e.g. bitmap image objects), or is of no interest for the collaborative work (so the internal structure is ignored), operations performed on internal elements can be simply treated as *Replacement* operations on these objects themselves. A *Replacement* operation consists of a *Delete* operation on the old version of the object, followed by an *Insert* operation for the new version of the object.[2] This is a useful and important data address adaptation technique for determining the data granularity of collaborative activities that can be merged by using OT.

A Tree of Linear Addressing Domains for a PowerPoint Document

Based on the address adaptation analysis in previous subsections, all data objects of a PowerPoint document can be mapped into a tree of linear addressing domains, as shown in Figure 3.4. The root node corresponds to the top level of the document and contains multiple independent linear addressing domains for the sequence of normal slides in the document, the *Slide Master*, the *Title Master*, the *Handout Master*, and the *Notes Master*, respectively. A second-level node corresponds to a slide and contains two independent linear addressing domains: one is for the sequence of graphic objects inside this slide, and the other is for the sequence of characters in the *Notes* editing area. A linear addressing domain in a particular node can be identified by a unique name within that node (determined by the PowerPoint API). A third-level node corresponds to a *Textbox* object and contains a single linear addressing domain.

[2] The new version of the object contains the effects of the operations performed on the internal elements of the object.

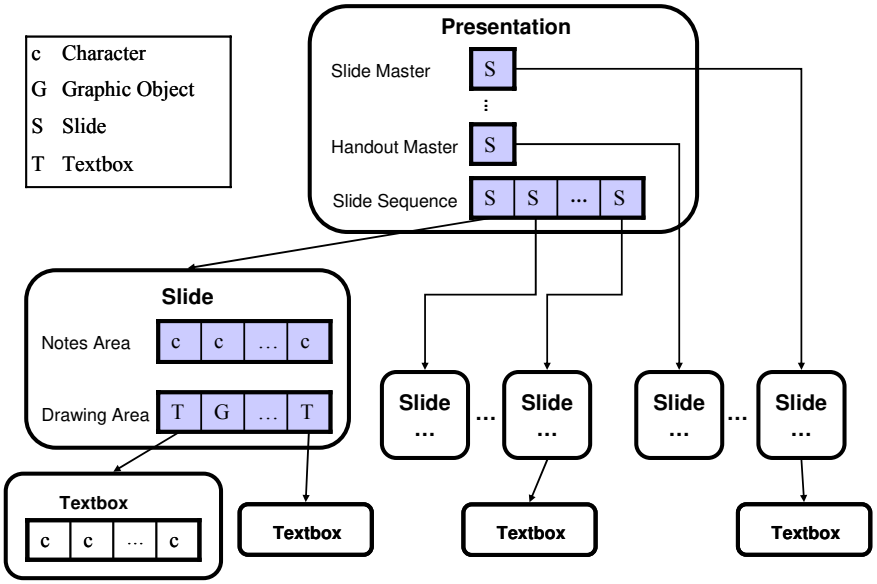


Figure 3.4 A tree of linear addressing domains for a PowerPoint document.

According to the data model in Figure 3.4, to access a normal slide in the top-level slides sequence, one pair of information pieces is needed: the unique name for the domain corresponding to the normal slides sequence, and the target slide's position reference (i.e. its sequence number). To access a graphic object in the graphic drawing area of a normal slide at the second-level, however, two pairs of information pieces are needed: the first pair contains the unique name for the domain corresponding to the slides sequence, and the target slide's position reference; and the second pair contains the unique name for the domain corresponding to the graphic drawing area in the slide and the target graphic object's position reference. To access a character object in a Textbox at the third level, three pairs of information pieces are needed: apart from the first two pairs for addressing the *Textbox* object, the third pair is to address the specific character in the *Textbox*.

As discussed above, data models of both Word and PowerPoint can be adapted to a tree of linear addressing domains (see Figure 3.2 and Figure 3.4). In unconstrained collaboration, concurrent editing operations generated by distributed users may target any linear addressing domain. The underlying OT

technique should be extended to support this data model. Issues related to extending the OT data model will be discussed in Chapter 4.

3.3. The Operation Model Adaptation

The objective of the operation model adaptation is to bridge the gap between operation models of the single-user application API and OT. In operation adaptation, the following issues must be addressed: how user-generated operations are intercepted, represented, and propagated among collaborating sites; how user-generated operations are processed by the OT technique for consistency maintenance; and how OT-processed operations are interpreted by the application's API for replaying their effects at remote sites. This section discusses operation adaptation-related issues and techniques learned from adapting the operation models of Word and PowerPoint.

3.3.1. The Adapted Operation

AO as the Vehicle for Representing and Propagating the User's Interaction

By means of the application's API, the user's interactions can be intercepted as a sequence of input events, such as *key-down*, *key-up*, and *mouse-move*. These input events, however, cannot be directly propagated to remote sites and replayed as-is. This is because, in an unconstrained collaboration environment, remote applications may be in different status and replaying the same sequence of input events on them may not achieve the desired effect. Moreover, there is no need to propagate all local input events to remote sites. For example, local input events that remote sites are not interested in (e.g. some window open/close events) may not need to be propagated. Most importantly, these low level events must be converted into high level operations in order to take advantage of OT for consistency maintenance. Therefore, the sequence of local input events needs to be filtered and converted into a sequence of semantically meaningful units, called *Adapted Operations* (AO). In this role, AOs serve as the vehicle for representing

the user's interactions with the application and for propagating the user's interactions among collaborating sites. Technical issues involved in AO generation will be discussed in Section 3.3.3.

AO as the Bridge between the API and OT

When an AO arrives at a remote site, it must first be processed by OT for consistency maintenance, and then be interpreted by means of the API for replaying its effect on the remote document. In this role, AOs act as the bridge between the API and underlying OT. With AOs residing between the API and OT, the task of operation adaptation between the API and OT is decomposed into two subtasks:

(1) AO-PO adaptation, which translates the AO into suitable *Primitive Operations* (PO) to be processed by OT; and
(2) API-AO adaptation, which interprets the AO by means of the API.

One approach to AO-PO adaptation is to extend the basic OT operation model to cover all AOs (i.e. treat every AO as a PO), so that every pair of AOs can be directly transformed by a specific OT function. If a single-user application supports *N* different data-manipulation AOs, then *N* * *N* different transformation functions are needed for adapting this application. A major problem with this approach is that application level transformation functions are too complex to design and to ensure correctness.[3] Another problem is that transformation functions defined for AOs are application-specific and not reusable in different applications.

Another approach, proposed in this work, is to extend the basic OT operation model with a new *Update* operation, and to translate application level AOs into three generic POs: *Insert*, *Delete*, and *Update*. The advantage of this approach is that the extended OT operation model becomes more powerful and capable of

[3] To get an idea about the complexity of designing two string-wise editing operations *Insert* and *Delete*, the reader is referred to Sun et al. (1998).

supporting word processing applications, and at the same time remains small and application-independent. The challenge with this approach is how to translate an AO into suitable POs so that applying OT on these POs can achieve the correct transformation effect on the AO itself. Technical issues involved in translating AOs to POs are discussed in Section 3.3.4. Issues and solutions involved in extending OT for supporting *Update* will be discussed in Chapter 4.

3.3.2. Defining AOs for Word and PowerPoint

Since AOs play a central role in bridging the gap between the operation models of the application API and the underlying OT technique, the definition of AOs for an application is a key aspect of operation adaptation for that application. In the following subsections, the data-related AOs defined for Word and PowerPoint will be briefly described.

Adapting Word Operations

The AOs defined for Word (Word-AO) are illustrated in Figure 3.5-(b). To facilitate the interpretation of Word-AOs by the Word API, it is essential for Word-AOs to carry the type information of the target data objects. This is because different types of data object are manipulated by different object methods in the Word API to achieve data-type-dependent editing effects. The strategy is to group and name Word-AOs according to the Word API data object types they are processing. These operation groups include: the *text* operation group (corresponding to the *Range* object in the Word API), the *inlineObj* operation group (corresponding to the *inlineShape* objects in the Word API), and the *floatingObj* operation group (corresponding to the *Shape* object in the Word API), etc. It should be pointed out that AOs are aware of data object types but need not be aware of the internal data structures of these types, which is the knowledge of the Word API implementation.

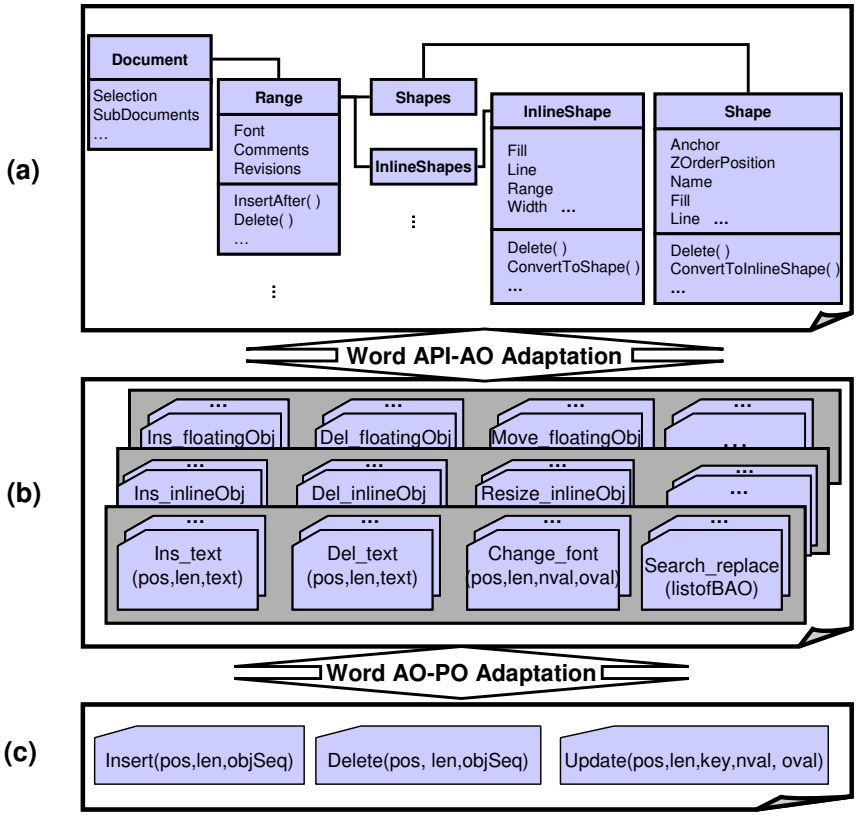


Figure 3.5 Three layers in Word operation adaptation.

At the OT layer, there are three primitive operations (Sun et al. 2004):

(1) *Insert(pos, num, objSeq)* denotes an *Insert* operation to create a sequence of *num* objects *objSeq* starting at position *pos* in the OT data model.

(2) *Delete(pos, num, objSeq)* denotes a *Delete* operation to remove a sequence of *num* objects *objSeq* starting at position *pos* in the OT data model.

(3) *Update(pos, num, key, nval, oval)* denotes an *Update* operation to change the attribute *key*, from old-value *oval* to new-value *nval*, of a sequence of *num* objects starting at position *pos* in the OT data model.

These three POs are generic in the sense that they are independent of object types (the *objSeq* parameter may refer to a sequence of characters, or graphics, etc.), attribute types (the *key* parameter may represent any object attribute like color, size, or position, etc.), and attribute values (the *nval* or *oval* parameter may

represent any attribute value). The OT layer does not need to know the object type, attribute type, or attribute value to do its work (Sun et al. 1998; Sun 2002b).

To facilitate the translation from Word-AOs to POs (shown in Figure 3.5-(c)), Word-AOs are also named and grouped in another dimension according to the three PO types: *Insert*, *Delete*, and *Update*. For example, for the text operation group, there are *Insert-text*, *Delete-text*, and *Change-font* (an *Update* for text), etc.; for the inline object operation group, there are *Insert-inlineObj*, *Delete-inlineObj*, and *Resize-inlineObj* (an *Update* for inline objects), etc.; and for the floating object operation group, there are *Insert-floatingObj*, *Delete-floatingObj*, and *Move-floatingObj* (an *Update* for floating objects), etc.

Word-AOs must carry information needed by the underlying OT to support group undo (Sun 2000; 2002b). For example, all delete operations carry one parameter for saving the deleted object (a text, inline, or floating object); and all update operations carry one extra parameter (denoted as *oval* in Figure 3.5-(b)) for saving the old attribute value before performing the update.

Adapting PowerPoint Operations

The PowerPoint API (Figure 3.6-(a)) models a PowerPoint document as a *Presentation* object. From the *Presentation* object, a *Slides* object can be accessed, which models the sequence of slides in the document. The *Slides* object contains various methods for creating *Slide* objects and accessing a particular slide (by slide-sequence). From the *Slide* object, a *Shapes* object can be accessed, which models the collection of graphic objects inside a slide. The *Shapes* object contains various methods for creating *Shape* objects and accessing an existing *Shape* object (by index-address or object-name).

The AOs defined for PowerPoint (PPT-AO) are illustrated in Figure 3.6-(b). To facilitate the interpretation of PPT-AOs by the PowerPoint API, PPT-AOs are named and grouped according to the PowerPoint API data object types they are processing. These groups include: the *slide group* (corresponding to the *Slide*

object in the PowerPoint API), and the *graphicObj group* (corresponding to the *Shape* object in the PowerPoint API), etc. On the other hand, to facilitate the translation from PPT-AOs to POs, PPT-AOs are also named and grouped in another dimension according to the three PO types. For example, for the *slide group*, there are *Insert-slide*, *Delete-slide*, and *Change-effect* (an *Update* for the *Slide* object), etc.; for the *graphicObj group*, there are *Insert-graphicObj*, *Delete-graphicObj*, and *Resize-graphicObj* (an *Update* for the *graphicObj* object), etc. Like Word-AOs, PPT-AOs also carry additional parameters required by the underlying OT for supporting group undo.

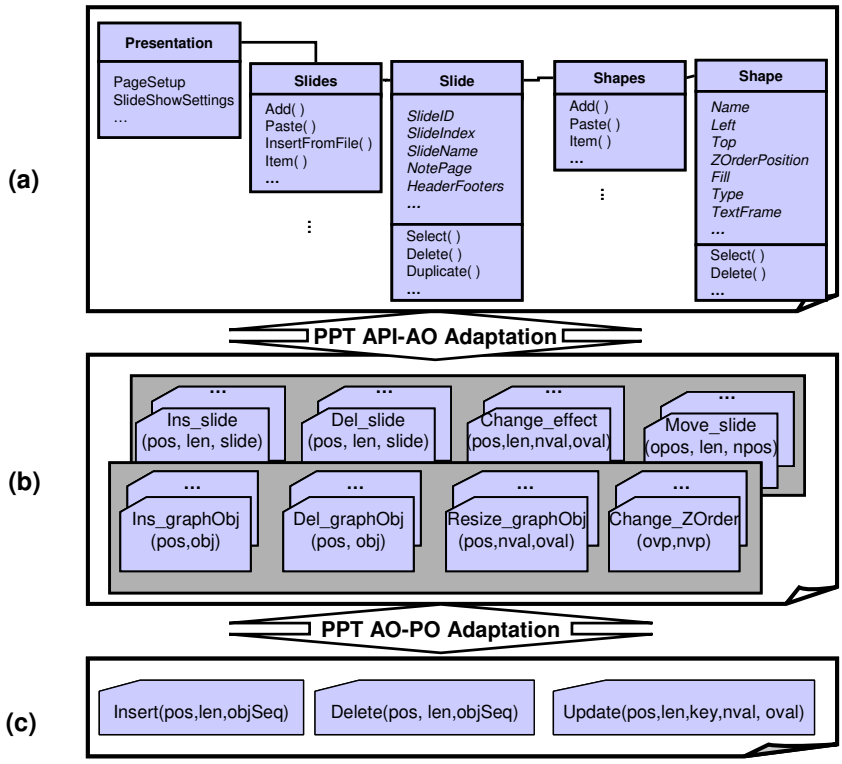


Figure 3.6 Three layers in PowerPoint operation adaptation.

3.3.3. Event Interception and AO Generation

The complexity of intercepting the user's interactions depends on the interface techniques adopted by the application, the operation types supported by the application, and the level and power of the API of the application and its execution environment.

In the current CoWord and CoPowerPoint systems, the user is restricted to using the keyboard and mouse to interact with the application. All user input events can be intercepted before they reach the application. The information available from the intercepted input events, however, is not sufficient to fully define an AO. To generate an AO, the application's API must be used to detect what object (e.g. text, inline, or floating) the user is accessing, to determine what operation (e.g. insert, delete, or update) the user is performing on the object, and to derive the parameters of this operation, including the position references of the object in data model, the inserted/deleted object, or the updated object attribute (both new and old values).

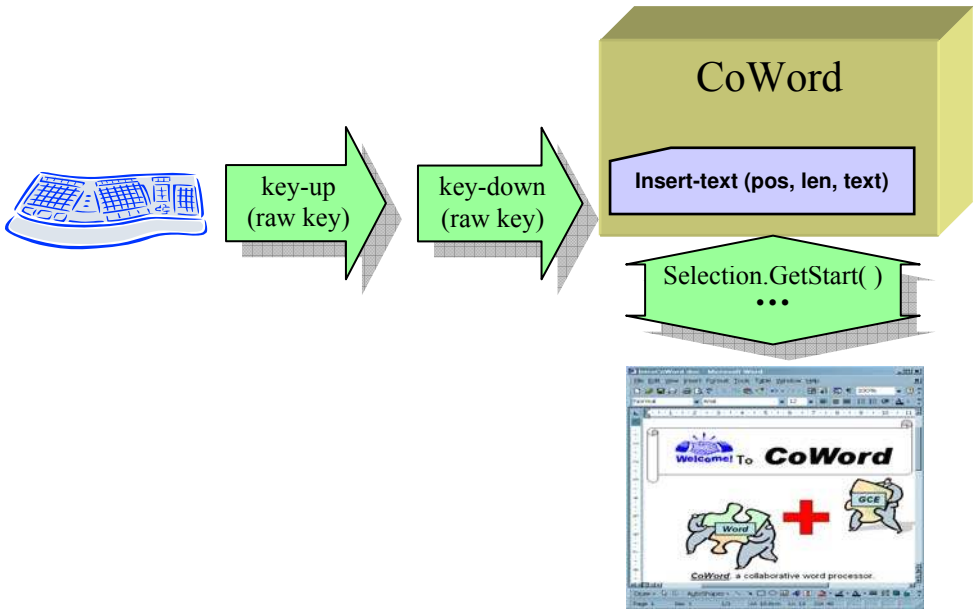


Figure 3.7 Intercepting keyboard events and generating the *Ins_Text* AO in CoWord.

Figure 3.7 shows an example of intercepting keyboard events and generating an *Insert-text* AO in CoWord. When the user inputs a character into the Word document from the keyboard, a pair of *key-down* and *key-up* input events will be generated and intercepted. Parameters of these events include some low-level information, such as the virtual code of the pressed key and the state information of some auxiliary keys. From the intercepted events, we know the user has inserted a character into the document, thus deriving that the AO type is *Insert-*

text. Then the Word API is called to get parameters of this *Insert-text* AO. For example, the position reference of this insertion is derived from the current cursor position, which can be obtained via the Word API *Selection.GetStart( )*. Also, the real effect (i.e. the formatted character) of this insertion can be obtained by calling other Word API functions. This formatted character, rather than the raw key code from the intercepted input events, is used as the *text* parameter of the *Insert-text* AO.

We must stress the importance of querying the application for the real effect of the user-inserted character and using the formatted character as the AO parameter. If we used the raw key code from the intercepted input events as the AO parameter for the inserted character, then it would be very difficult to correctly determine its full effect at remote sites in the presence of concurrency. This is because the determination of the full effect (e.g. font, size, color) of a character inserted from the keyboard is dependent on the context (i.e. the existing characters surrounding the newly inserted character). Rather than re-inventing Word's internal functionalities, we let Word do the real work (in determining the full effect of the user's interaction), and then query Word for the final effect and carry this effect as an AO parameter to remote sites for replay.

Another issue is the timing of querying the application for deriving AO parameters. Since the user's input events are intercepted before they reach the application, we can derive AO parameters by querying the application before and/or after the local execution of the user's input events:

(1) For creation operations (e.g. *Insert-slide*(*pos, num, slide*)), the created object (slide) can be obtained after the local execution.
(2) For deletion operations (e.g. *Delete-slide*(*pos, num, slide*)), the deleted object (slide) must be obtained before the local execution.
(3) For update operations (e.g. *Resize-GraphicObj*(*pos, nval, oval*)), the parameter *oval* (the old size of the graphic object) must be obtained before

the local execution, but the parameter *nval* (the new size of the graphic object) can be obtained after the local execution.

Finally, the functional knowledge of the application (from the user's point of view) also plays an important role in the process of understanding the user's interaction and generating the AO. For example, when the user selects a range of characters and then clicks the "Bold" button, we (the programmers) know that the user must have generated an *Update* operation on the selected characters according to the application's function from the user's point of view. Moreover, the functional knowledge of the application is also important in determining whether or not a user-level operation should be converted to an AO. In Word, for example, the user may perform a local *Copy* operation on a selected object. From the functional knowledge of Word, we know that a *Copy* operation creates a copy of the selected object in the local clipboard buffer but has no effect on the document state, so it need not be converted into an AO. When the user later performs a *Paste* operation, a previously copied object in the local clipboard (determined by Word) will be inserted into the document. At this moment, one *Insert* AO can be generated and propagated to remote sites. In effect, a pair of user-level *Copy* and *Paste* operations are merged into a single *Insert* AO according to their combined effect on the document state. This solution is simple and clean because it does not require any change to the execution of local *Copy* and *Paste* operations or any additional mechanisms for supporting remote *Copy* and *Paste* operations. If *Copy* and *Paste* were represented as separate AOs, then not only would there be a need to devise additional mechanisms for treating *Copy* as a special read AO, but we would also have to maintain consistent clipboard buffers (in addition to consistent document states) at all sites.

In summary, information from the following three sources are needed in generating AOs: (1) the user's interactions (intercepted by the application API), (2) the effect of the user's interaction on the application state (queried from the application's API), and (3) the functional knowledge of the application (obtained from usage experience or the application's user manual).

3.3.4. AO-PO Adaptation

The task of AO-PO adaptation is to translate an AO into suitable POs for OT processing. The term *OT-relevant parameters* is used to mean those AO parameters that may be affected by concurrent operations, such as the position references of an AO (including the *pos* and *num* parameters). The following two criteria have been used as guidelines to determine what POs should be used to represent a given AO.

(1) The OT-relevant parameters of the AO must be fully represented by the POs.
(2) The impact of the AO on the OT-relevant parameters of other concurrent AOs must be fully captured by the POs.

An AO is called a basic AO if it can be represented by a single PO, or a compound AO if it must be represented by multiple POs. For example, an AO for creating an object, removing an object, or changing an attribute (e.g. color, font style, or size) of an object, is a basic AO since its OT-relevant parameters and its impact on other concurrent AOs can be fully captured by a single PO. The translation from a basic AO to a single PO is straightforward: the PO-type information in the AO's name (see Figure 3.5-(b) and Figure 3.6-(b)) can be used to determine the type of the PO (i.e. *Insert*, or *Delete*, or *Update*); the OT-relevant parameters (e.g. position references) of the AO can be directly used in the PO. For example, an *Insert* AO (e.g. *Insert-text*, *Insert-inlineObj*, or *Insert-floatingObj*) can be translated into an *Insert* PO, whose *pos* and *length* parameters are taken directly from the AO, but whose *objSeq* parameter is just a reference to a generic object – the real object type (text, inline, or floating) and internal structure of the data object are of no interest to OT.

On the other hand, an AO for moving one character from position X to position Y (in CoWord) is a compound AO since it has to be translated into two POs: a *Delete* operation representing the deletion of the character at position X, and an *Insert* operation representing the insertion of the deleted character at position Y.

Another example compound AO is the *Search-and-Replace* operation, which must be represented by a sequence of *Delete* and *Insert* PO pairs. Moreover, the user may select a collection of disjoint objects (e.g. floating graphic objects in Word, or slides in PowerPoint), and apply a single operation (e.g. deletion or update) on them. This single user-level operation can be expressed as a single AO, but this AO has to be treated as a compound AO since no single PO is able to identify multiple disjoint objects.

In the above compound AO examples, the relationship between the compound AO and its representing POs is obvious, but this is not always the case. In Word, for example, the user can insert a new comment into the document, which is represented as a single *Comment-insert* AO in CoWord. The overall effect of this AO on the document consists of highlighting (with a color) the selected text segment, and creating a comment element in the *Comment Story* (an editing area independent of the main document). This AO carries, among others, three OT-relevant parameters: (1) the starting position of the selected segment, (2) the length of the selected segment, and (3) the insertion position of the comment element in the *Comment Story*. These parameters are OT-relevant since they may be changed by and have impact on other concurrent operations. This *Comment-insert* AO is a compound operation since no single PO is able to represent all three parameters and to capture its impact on other concurrent operations. Based on the two criteria for AO-PO adaptation, this compound AO can be translated into two POs: one *Update* operation (*Highlighting*) for representing parameters (1) and (2), and one *Insert* operation for representing parameter (3) and its impact on other concurrent AOs. It must be pointed out that these POs are involved in OT processing only, not in the API interpretation of the AO (see Section 3.3.6).

The types of compound AO and the methods of translating compound AOs into POs are application-specific. Techniques for adapting complex compound AOs will be discussed with examples of collaborative table editing and collaborative graphic object grouping in Chapter 5.

After translating an AO into suitable POs, OT will be applied to these POs. Then, these transformed POs will be used to update the corresponding OT-relevant parameters of the AO. In this way, the AO is effectively transformed by OT. Therefore, AO-PO adaptation can be regarded as an application-specific extension to the OT operation model.

3.3.5. AO-API Adaptation

The task of API-AO adaptation is to interpret the transformed AO by means of the API. The interpretation of a basic AO is straightforward: the data type information (e.g. text, inline, or floating object) encoded in the AO name (see Figure 3.5-(b) and Figure 3.6-(b)) is used to determine suitable API object class types for the target object; the position references of the AO are used to find out the target object in the document; other parameters of the AO are used in the API method calls in order to replay the AO's effect on the document.

Some compound AOs are composed of a list of basic AOs since the effects of these compound AOs are achieved by sequentially executing these basic composing AOs. The interpretation of these AOs can be achieved by sequentially interpreting the basic composing AOs as well. For example, a *Search-and-Replace* AO is composed of a list of basic *Delete* and *Insert* AO, which are determined at the local site. When this compound AO arrives at a remote site, all composing basic AOs are first translated into corresponding POs and processed by OT in the AO-PO adaptation phase. Then, in the API-AO adaptation phase, all transformed basic composing AOs are interpreted one by one to achieve the effect of the compound AO.

However, not all compound AOs are composed of multiple basic AOs. As discussed in Section 3.3.4, the *Comment-insert* AO is a compound AO since it has to be translated into two POs (one *Update* plus one *Insert*) for the purpose of OT. This compound AO, however, is not composed of a basic *Update* AO (to highlight the selected text segment) and a basic *Insert* AO (to insert the comment

element into the document) since the effect of inserting a comment into the document cannot be achieved by sequentially executing these two basic AOs. In fact, the Word API provides a special method to insert a comment into the document. Therefore, the interpretation of the *Comment-insert* compound AO is achieved by invoking a single Word API method. This example highlights the independency of the API interpretation and the PO translation: the API interpretation of an AO is based on the semantics of this AO, which is not related to the POs that represent the AO for the purpose of OT.

The relationship between AO-PO adaptation and API-AO adaptation can be summarized as follows: the former is responsible for getting the AO's parameters (syntax) right in the presence of concurrency; the latter is responsible for getting the AO's execution effect (semantics) right under the current application context. Because of this division of responsibilities, POs (and OT) do not require the awareness of the semantics of AOs, and the API interpretation does not need to worry about concurrency.

3.4. Summary

In this chapter, an innovative *Transparent Adaptation* (TA) approach which can be used to convert single-user applications into collaborative ones without changing the source code of the original application, has been discussed.

The TA approach is based on (1) the use of the single-user application's API to intercept and replay the user's operations, and (2) the use of *Operational Transformation* (OT) to manipulate intercepted user operations for supporting responsive and unconstrained (i.e. concurrent and free) multi-user interactions with the shared application. For this approach to work, the shared application's API needs to be adapted to the data and operation models of the OT technique. Two TA-based systems, CoWord and CoPowerPoint, were used as examples to discuss data and operation model adaptation techniques.

The user's view of a Word document does not look like a linear sequence of objects, but from Word API's view, all objects, including characters, inline objects and floating objects, can be accessed by their positional references in a linear addressing space. Taking other auxiliary document elements (e.g. Comments, Headers, Footers) into account, the whole Word document can be modeled as a tree of linear addressing domains. Similarly, with the PowerPoint API, all data objects in a PowerPoint document can also be accessed with positional references in a tree of linear addressing domains. To adapt the data models of Word and PowerPoint, the OT data model should be extended correspondingly.

In the TA approach, user input events are converted into semantically meaningful *Adapted Operations* (AO) for representing and propagating the user's interaction. Moreover, AO is also the bridge between OT and the API, so the operation model adaptation task is decomposed to (1) AO-PO adaptation, which translates the AO into suitable *Primitive Operations* (PO) to be processed by OT, and (2) API-AO adaptation, which interprets the AO by means of the API to replay the user's interaction. For the purpose of AO-PO adaptation, the OT operation model should be extended to support a new *Update* operation.

To adapt the operation models of Word and PowerPoint, a set of Word-AOs and PPT-AOs are defined respectively. To facilitate the interpretation of Word/PPT-AOs by the API, Word/PPT-AOs are named and grouped according to the data object types they are processing. In another dimension, to facilitate the translation from Word/PPT-AOs to POs, AOs are also named and grouped according to the three PO types.

To generate AOs in response to the user's interaction, three sources are needed: (1) the user's interaction intercepted by the application's API, (2) the effect of the user's interaction on the application state queried via the application's API, and (3) the functional knowledge of the application. However, to perform AO-PO adaptation and API-AO adaptation for different AO, different strategies are

needed. For basic AOs, both AO-PO adaptation and API-AO adaptation are straightforward, but adaptation methods for compound AOs are more complex: application- and operation-specific methods are needed.

Chapter 4

Extending Operational Transformation for Supporting TA

As discussed in Chapter 3, leveraging single-user applications into multi-user collaborative versions based on the TA approach requires extensions to both the data model and the operation model of the basic OT technique. In the data model aspect, the OT technique should be extended to support the data model based on a tree of linear addressing domains; in the operation model aspect, the OT technique should be extended to support a new operation type, *Update*. This chapter discusses these two extensions to the basic OT technique.

4.1. Introduction

The OT technique consists of two layers: high-level transformation control algorithms and low-level transformation functions (see Chapter 2). When the data and operation models of the OT technique are extended, the transformation control algorithm needs no change, because it is independent of the addressing schemes and operation types. Changes should be done at the transformation function level, because they are related to the addressing schemes and operation types. Existing transformation functions in the basic OT technique are capable of handling *Insert* and *Delete* operations based on a single linear addressing domain only. Therefore, our strategy is to extend the transformation functions so that they

can handle transformation of all three operations on the extended data model. At the same time, the high-level control algorithms are kept unchanged.

The rest of this chapter is organized as follows. The extension to the OT data model is discussed in Section 4.2. The extension to the OT operation model for supporting *Update* is discussed in Section 4.3. Finally, this chapter concludes with a summary in Section 4.4.

4.2. Extending the OT Data Model

4.2.1. Extending the OT Data Model

XOTDM: an eXtended OT Data Model

To meet the need for supporting complex data models, such as those of Word (Figure 3.2) and PowerPoint (Figure 3.4), the basic OT data model should be extended from a single linear addressing space to a tree of addressing groups, where each group consists of multiple independent linear addressing domains, as shown in Figure 4.1. XOTDM is used as the name of this eXtended OT Data Model. Inside each addressing group, independent linear addressing domains are identified by their unique names within that group. A data object is mapped to a position in a linear addressing domain only if it has the position number as its address in this domain. A data object is a terminal object if it has no internal data structure or its internal data structure is not addressable. A data object is an intermediate object if it has an addressable internal data structure. In XOTDM, a terminal object has no link out of it, but an intermediate object has a link leading to a lower level addressing group, which represents this object's internal addressing space.

Data objects of a wide range of different types of document can be mapped onto XOTDM. For example, all characters in a plain text document can be mapped into a tree of a single addressing group, which contains a single linear

addressing domain. All data objects in this domain are terminal objects since plain text characters have no internal structure.

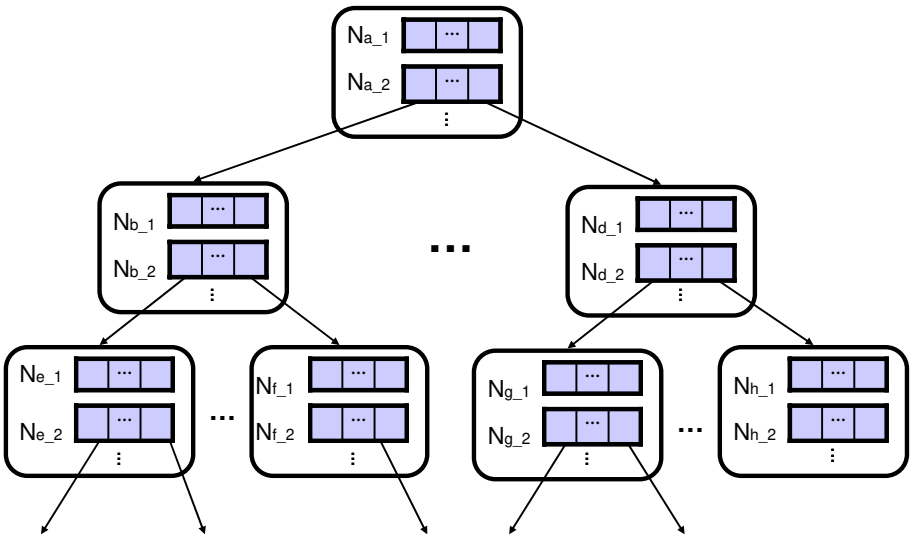


Figure 4.1 The XOTDM tree: an eXtended OT Data Model.

Data objects in a Word document can be mapped into a two-level XOTDM (compare Figure 3.2 with Figure 4.1). The top-level addressing group contains a single linear addressing domain, corresponding to the range of data objects in the main body of the document. Characters and graphic objects without addressable internal structures (or where the internal structure is of no interest) are terminal objects. Graphic objects with addressable internal structures, *Comments*, *Notes*, and *Section-Breaks* are intermediate objects which have links to addressing groups at the second level. A second-level addressing group for a *Comment* or *Notes* contains a single linear addressing domain, corresponding to the sequence of characters in the comments or notes; a second-level addressing group for a *Section-Break* contains multiple linear addressing domains, corresponding to the multiple independent sequences of data objects in the *Headers* and *Footers* associated with the section.

Data objects of a PowerPoint document can be mapped into a three-level XOTDM as well (compare Figure 3.4 with Figure 4.1). The top-level group contains multiple independent linear addressing domains, corresponding to the

sequence of normal slides in the document, and various master slides. All data objects in the top-level group are intermediate objects since they represent slides which have addressable internal data structures. A second-level addressing group corresponds to the internal addressing space of an individual slide, with two independent linear addressing domains: one is for the sequence of graphic objects in the drawing area, and the other is for the sequence of characters in the notes area. Data objects in the notes area are all terminal objects. Textboxes in the drawing area are intermediate objects since they have addressable internal data structures (represented by third-level nodes). All other data objects in the drawing area are treated as terminal objects because either they have no addressable internal structure or their internal structures can be ignored. All data objects in a third-level node (representing a *Textbox* node) are treated as terminal objects.

It should be stressed that XOTDM reflects only the relationships of data object addresses, rather than data objects themselves. Data objects in an application may have arbitrarily complex relationships, which cannot and need not be mapped into XOTDM for the purpose of applying OT. Two data objects are mapped into two adjacent positions of a linear addressing domain in XOTDM just because they have adjacent positional addresses in this domain. Their dynamic positional relationship in the addressing domain is independent of their static relationship in the object class hierarchy and is independent of their visual relationship on the user interface.

Addressing Data Objects

Under XOTDM, a data object inside a given addressing group can be uniquely addressed by a pair *(n, p)*, where *n* is the name of a linear addressing domain in this group, and *p* is the object's position in this domain. To address any data object in an XOTDM, a vector of *(n, p)* pairs is needed:

$$vp = [(n_0, p_0), (n_1, p_1), \ldots, (n_i, p_i), \ldots, (n_k, p_k)]$$

where $vp[i] = (n_i, p_i)$, $0 \le i \le k$, represents one addressing point at level i.

For a Word document, a vector of two (*n, p*) pairs can be used to identify a data object at the main document layer (addressed by the first pair), and a data object inside an intermediate object (addressed by the second pair). For example, to perform an operation in the main document, the editing operation needs a vector of only one (*n, p*) pair: *vp* = [("Main Text", p_0)], where p_0 refers to the target object's linear position in the main document. To create a data object in a *Header* associated with a *Section-Break* in the main document, however, the editing operation should carry a vector of two (*n, p*) pairs: *vp* = [("Main Text", p_0), ("Header-1", p_1)], where p_0 is the position of the *Section-Break* link in the main document, and p_1 is the position of the created data object in the linear addressing domain named as "Header-1".

For a PowerPoint document, a vector of two (*n, p*) pairs can be used to identify a slide at the top slides sequence level or in a master slide (addressed by the first pair), and a graphic object (in the drawing area) or a character (in the notes editing area) inside this particular slide (addressed by the second pair). For example, to insert or delete a slide in the "slide-sequence" at the top level, the editing operation needs a vector of one (n, p) pair, such as *vp* = [("slide-sequence", 2)] refers to slide "2" in the "slide-sequence" domain. To update a graphic object in a normal slide, the editing operation should carry a vector of two (*n, p*) pairs, such as *vp* = [("slide-sequence", 1), ("drawing area", 3)] refers to the graphic object at position "3" in the "Drawing Area" of slide "1" in the "slide-sequence" domain. To insert a character object in a Textbox in a normal slide, the editing operation should carry a vector of three (*n, p*) pairs, such as *vp* = [("slide-sequence", 1), ("drawing area", 2), ("Textbox", 3] refers to the character object at position "3" in the "Textbox", which is the number "2" graphic object in the "drawing area" of slide "1" in the "slide-sequence" domain.

4.2.2. Target-Domain Relationships among Operations

As discussed earlier, to extend the OT data model to support XOTDM, changes to the OT technique are to be done at the transformation function level. To achieve this goal, one strategy is to redefine existing transformation functions so that they become capable of handling operations with vector addresses, as is done in Davis et al. (2002). The problem with this strategy is that all existing transformation functions have to be revised and re-tested, which is nontrivial. The strategy adopted in this research is to encapsulate the impact of the vector of (*n, p*) pairs in a wrapper vector-based OT function (the *VOT* ( ) function), but to keep all existing transformation functions unchanged. This strategy allows us to localize the impact of XOTDM and maximize the reuse of existing algorithms and functions.

Under XOTDM, the target data object of an operation must fall into one particular linear addressing domain. This particular domain is called the *target-domain* of this operation. The target-domain relationship among operations is very important in determining whether and how operations should be transformed against each other. For convenience of discussion, the following terminologies are introduced. Domain A is an *ancestor-domain* of domain B if there is a sequence of arrows from A to B in the XOTDM tree. The sequence of domains from the root domain to the target-domain (inclusive) of an operation is called the *domain-path* of this operation.

In an unconstrained collaborative editing session, multiple users may generate concurrent operations in the same or different target-domains. Concurrent operations in the same target-domain (e.g. O_n and O_2 performed on slide-1 in Figure 4.2) should be transformed against each other. This is because the execution of one operation in its target-domain may have impact on the position references and other parameters of concurrent operations in the same target-

domain. Existing OT functions for a single linear addressing domain can be directly used to transform operations performed on the same target-domain.

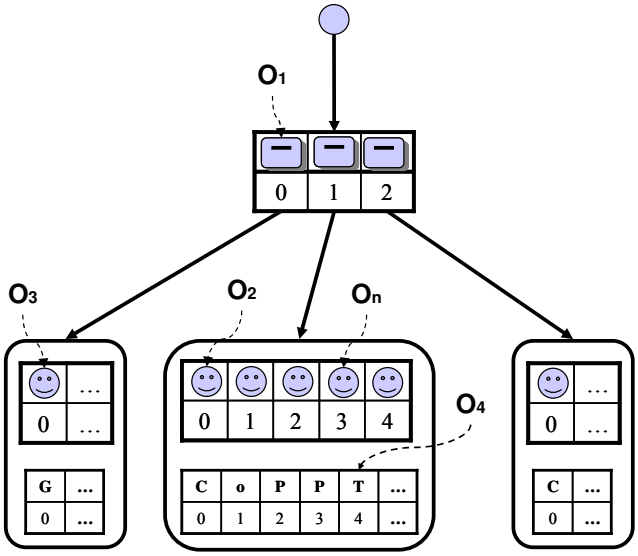


Figure 4.2 Concurrent operations in multiple domains of a CoPowerPoint document.

The question is whether concurrent operations in different target-domains need to be transformed against each other. The answer is yes and no, depending on the relationship between the target-domains of these operations. Given two concurrent operations O_n and O_x in different target-domains, if O_x's target-domain is an ancestor-domain of O_n's target-domain, and O_x is executed before O_n, then O_n must be transformed against O_x since O_x's execution in the ancestor-domain may have changed O_n's corresponding position reference. For example, in Figure 4.2, O_1 is performed on the root domain which is an ancestor of O_n's target-domain. If O_1 is executed before O_n, O_n must be transformed against O_1 since the execution of O_1 may change O_n's slide-sequence-number, which is a part of O_n's address. It should be pointed out that existing OT functions cannot be directly used to transform O_n against O_1, and a new function for transforming operations on different domains is needed (to be discussed later).

However, if the target-domain of O_x is not an ancestor-domain of O_n, then O_n need not be transformed against O_x since the execution of O_x may not have any

impact on O_n. For example, in Figure 4.2, O_n in slide-1 does not need to be transformed against another operation O_3 in slide-0, or against operation O_4 in the notes area of slide-1, since the target-domains of both O_3 and O_4 are not on the domain path of O_n.

To summarize, given two concurrent operations O_n and O_x and supposing O_x is executed before O_n, O_n needs to be transformed against O_x under the following two circumstances:

(1) O_x and O_n have the same target-domain; and
(2) O_x's target-domain is an ancestor-domain of O_n's target-domain.

In both cases, O_x's target-domain is one of the domains on the domain-path of O_n. This target-domain relationship between operations is called domain-dependence. A more precise definition of this relationship is given below.

Definition 4.1. Domain-dependence relation "$O_a \xrightarrow{d} O_b$". Given two operations O_a and O_b. Let D_a and D_b denote the target-domains of O_a and O_b, respectively. O_b is domain-dependent on O_a, denoted as $O_a \xrightarrow{d} O_b$, if D_a is one of the domains on the domain-path of D_b.

If D_a is not on the domain-path of D_b, then O_b is not domain-dependent on O_a, denoted as $O_a \not\xrightarrow{d} O_b$.

4.2.3. Checking Target-Domain Relationships

Like the concurrency relationship among operations, the domain-dependence relationship among operations is an essential condition in determining whether two operations need to be transformed. Also like the vector of operation counters (i.e. the state vector (Ellis and Gibbs 1989; Sun et al. 1998)) used for checking the

concurrency relationship among operations,[4] the vector of *(n, p)* pairs is used to check the domain-dependence relationship among operations.

```
Check_target_domain_relation(On, Ox)
{
        if(Domain_dependent(On, Ox))
                if(|On.vp| == |Ox.vp|);          //|On.vp| is the vector's length
                        return SAME_DOMAIN;
                else
                        return ANCESTOR_DOMAIN;
        else
                return INDEPENDENT_DOMAIN;
}

Domain_dependent(On, Ox)
{
        if(|On.vp| < |Ox.vp|)
                return false;
        for(int i = 0; i < |Ox.vp| - 1; i++)
                if(On.vp[i] != Ox.vp[i])          //Note: vp[i] is a (n, p)-pair
                        return false;
        if(On.vp[|Ox.vp| - 1].n != Ox.vp[|Ox.vp| - 1].n)
                return false;
        return true;
}
```

Figure 4.3 Checking the target-domain relationship.

The *Check_target_domain_relation( )* function in Figure 4.3 has been devised to check the target-domain relationship of O_n against O_x based on the vectors of *(n, p)* pairs (i.e. their *vp* address parameters). The *Domain_dependent(O_n, O_x)* function checks whether O_n is domain-dependent on O_x (i.e. $O_x \xrightarrow{d} O_n$). If O_n is domain-dependent on O_x, it is further differentiated whether the two operations have the same target-domain (return SAME_DOMAIN), or O_x's target-domain is an ancestor-domain of O_n's target-domain (return ANCESTOR_DOMAIN). This differentiation is necessary because different transformation functions will be used for transforming O_n against O_x in these two sub-cases (see the *VOT*( )

[4] The concurrency relationship among operations is checked by high level transformation control algorithms based on operations' state vector time-stamps (Sun et al. 1998; Sun 2002b).

function in Figure 4.4). If O_n is not domain-dependent on O_x, then INDEPENDENT_DOMAIN is returned.

4.2.4. The VOT function

```
VOT(On, Ox)
{
        switch(Check_target_domain(On, Ox)) {
                case SAME_DOMAIN:
                        Transform_same_domain(On, Ox);
                        break;
                case ANCESTOR_DOMAIN:
                        Transform_ancestor_domain(On, Ox);
                        break;
                case INDENPENDENT_DOMAIN:
                        break;        // do nothing
        }
}

Transform_ancestor_domain(On, Ox)
{
        last = |Ox.vp| - 1;
        switch(Ox.type) {
                case Insert:
                        if(Ox.vp[last].p <= On.vp[last].p)
                                On.vp[last].p++;
                        break;
                case Delete:
                        if(Ox.vp[last] < On.vp[last])
                                On.vp[last].p--;
                        else if(Ox.vp[last] == On.vp[last])
                                SetNULL(On);
                        break;
                case Update:
                        break;        //do nothing
        }
}
```

Figure 4.4 A wrapper OT function for transforming operations with vector addresses.

As described in Chapter 2, the transformation control algorithm of an OT technique determines which operation should be transformed against other operations, and then calls the transformation functions to do the transformation. In particular, when a new operation from a remote site arrives, the control algorithm (e.g. GOTO (Sun and Ellis 1998)) scans (and may also reorder) the

history buffer of executed operations, and selects operations to transform against the new one.[5] To transform the new operation against those in the history buffer, the control algorithm calls the transformation functions (i.e. IT and ET).

However, to transform an operation O_n against a concurrent operation O_x defined in XOTDM, we do not directly call the IT or ET functions. Instead, the *VOT( )* function in Figure 4.4 is called. In the *VOT( )* function, the *Check_target_domain_relation( )* function is first called to differentiate the three kinds of target-domain relationship: SAME_DOMAIN, ANCESTOR_DOMAIN or INDEPENDENT_DOMAIN, between O_n and O_x, based on their *vp* parameters. Then three different transformation cases are handled separately.

First, if both O_n and O_x have the same target-domain, then the execution of O_x may have impact on O_n's last position and other parameters (e.g. attribute values). In this case, O_n can be transformed against O_x by using transformation functions based on singular positions. This is achieved by calling the *Transform_same_domain( )* function, which encapsulates the conversion between vector positions and singular positions, and the invocation of existing transformation functions.[6]

Second, if O_x's target-domain is an ancestor of O_n's target-domain, then the execution of O_x may have impact only on O_n's corresponding position, not on O_n's attribute value parameters. In this case, O_n must be transformed against O_x by a new function *Transform_ancestor_domain( )*. The transformation result is dependent on O_x's type and the relationship between O_x's last position and O_n's corresponding position (which is the position with the same index as O_x's last position). If O_x is an *Insert* and its last position is smaller than or equal to O_n's

[5] For details of the GOTO algorithm, the reader is referred to Sun and Ellis (1998).

[6] Details of vector versus singular positions conversion are omitted for conciseness. For definitions of transformation functions based on singular positions, the reader is referred to Sun et al. (1998); Sun and Ellis (1998); Sun et al. (2004).

corresponding position, then O_n's position is incremented by one.[7] If O_x is a *Delete* operation and its last position is smaller than O_n's corresponding position, then O_n's position is decremented by one; but if these two positions are equal, which means O_n's target-domain has been removed by O_x, then function *SetNULL(O_n)* is invoked to set O_n to *NULL*.[8] If O_x is an *Update* operation, no change is made to O_n's position. It should be highlighted that O_n's type has no influence on the transformation result, which is a major difference between *Transform_ancestor_domain( )* and existing transformation functions defined for singular positions (Sun et al. 1998).

Third, if O_n is not domain-dependent on O_x, then the execution of O_x cannot have any impact on O_n. In this case, O_n is returned without any change.

Finally, it should be pointed out that although CoWord and CoPowerPoint use a vector of maximum two or three *(n, p)* pairs, the *VOT( )* function supports vectors of any number of *(n, p)* pairs.

4.2.5. Other Tree-Based OT Techniques

The extension of the basic OT technique to support a tree of multiple linear addressing domains represents an important advancement from previous work (Davis et al. 2002). The tree-based document modeling and vector-based addressing scheme have already been discussed in the context of XML-based documents in Davis et al. (2002), but the discovery of the tree-based document modeling and vector-based addressing in the Word and PowerPoint APIs is a valuable research finding. This finding is significant because it reveals the excellent match between the OT technique and a wide range of existing commercial off-the-shelf single-user applications, and thus greatly increases the applicability of the OT technique.

---

[7] For simplicity, it is assumed that each operation targets only one object. In other words, the value of its *num* parameter is always *one*.

[8] *NULL* is an empty operation without any effect on the document or in transformation.

Technically, the extension of the basic OT technique to the tree-based OT technique was achieved by embedding the vector positional references in all existing transformation functions in Davis et al. (2002). In contrast, the extension in this work encapsulates the impact of the vector-based addressing and transformation inside a wrapper transformation function (*VOT*( )), and keeps existing OT control algorithms and transformation functions unchanged. Above all, the tree-based OT technique presented in this chapter is the only one fully implemented and tested (in the CoWord and CoPowerPoint systems).

There is another tree-based OT technique, TreeOPT (Ignat and Norrie 2003), which is very similar to Davis et al. (2002). Some specific points of the TreeOPT algorithm include: the use of a tree of history buffers to reduce the number of transformations (at the cost of maintaining an explicit tree of buffers), and the use of an artificial zero-length *Delete* operation to represent an operation when it is transformed against other operations at high layers in the tree (which is a trick to get around the problem of the lack of knowledge of vector-based addressing in the basic OT technique).

The data models used in Davis et al. (2002) and Ignat and Norrie (2003) can be regarded as special cases of XOTDM, in the sense that there is only a single linear addressing domain in each addressing group.

4.3. Extending OT for Supporting Update

After the *Update* operation is introduced into the OT operation model, the OT technique supports three primitive operations: *Insert*, *Delete* and *Update* (see Chapter 3).

For supporting *Update*, OT needs to be extended with a set of *Update*-related transformation functions. However, the central issue in supporting *Update* is conflict resolution. This is because, in an unconstrained collaborative

environment, users may concurrently update the same attribute of a common object, resulting in conflicts. Corresponding techniques embedded in transformation functions are needed for conflict resolution and preservation of user's effort.

In GRACE (Chen 2001; Sun and Chen 2002), a *Multi-Versioning* (MV) technique was devised to preserve all operations' effects in the face of conflicts. With this technique, multiple versions of the same object are created to accommodate the effects of multiple conflicting *Updates*. This MV technique provides users with a complete picture about what other users intended to do in the situation of conflict, so that they could better assess the situation and react accordingly.

Due to the differences between the frameworks of GRACE and OT, directly applying the GRACE MV technique in OT for conflict resolution could significantly complicate the OT framework. In this research, a new MV technique, called *Multi-Version Single-Display* (MVSD) has been devised. The basic idea of MVSD is the following: when an object is updated by conflicting operations, multiple versions of the target object will be created and maintained internally (similar to GRACE), but only one version is displayed at the user interface (different from GRACE). Moreover, all versions of an object can be displayed (one by one) by invoking an AnyDisplay algorithm (Sun 2004).

The major merit of MVSD is that it fits very well in the OT framework. Furthermore, it naturally matches the interface features of existing single-user applications (e.g. MS Word). For details about supporting *Update* in OT and the MVSD technique, the reader is referred to Sun (2004).

4.4. Summary

This chapter has discussed techniques for extending the basic OT technique in two aspects. On the one hand, the data model of the basic OT technique has been

extended to support the XOTDM (eXtended Operational Transformation Data Model). On the other hand, the OT operation model has been extended to support a new operation, *Update*. With these two extensions, the TA approach is applicable to a wider range of single-user applications.

After the data model of OT is extended to a tree of linear domains, operations involved in the transformation may have different target-domain relations. Based on the target-domain relation definition, the solution to transforming operations in XOTDM was designed in two steps. First, an algorithm is designed to check the target-domain relation of two operations according to their position parameters. Then, a transformation function is designed to transform two operations according to their target-domain relation. To keep existing transformation functions unchanged, the impact of the vector-based addressing and transformation is encapsulated inside a wrapper transformation function. This extension to the OT technique has increased the capability of the OT technique to support collaboration on complex data structures (e.g. the Word and PowerPoint documents).

To extend the OT data model to support the *Update* operation, the first task is to design transformation functions that transform *Update* against other operations. Unlike *Insert* and *Delete*, *Update* operations may conflict with each other. Therefore, a conflict resolution technique must be designed. The multi-versioning strategy is ideal for conflict resolution due to its ability to preserve all users' intensions in the face of conflict, but exiting the multi-versioning technique cannot be directly applied in the OT framework. To achieve the multi-versioning effect with OT, a Multi-Version Single-Display (MVSD) strategy is adopted in this research. In the face of conflict, multiple versions of the target object of conflict *Update* operations are maintained internally, but only one version is displayed in the user interface. Moreover, an AnyDisplay algorithm is able to display any version of an object.

Chapter 5

Applying TA to Complex Application Data Structures and Operations

In Chapter 3, a basic TA approach which can be used to adapt common rich format text and graphics editing in single-user applications, has been presented. However, many off-the-shelf commercial single-user applications have complex data structures and editing functionalities that cannot be directly adapted by the techniques presented in Chapter 3. To support these complex data structures and editing functionalities in collaborative versions, special adaptation techniques need to be designed in the TA framework.

In this thesis work, two special adaptation techniques – CoTable and CoGroup – have been devised to support collaborative table editing and collaborative graphic object grouping, respectively. The reasons for targeting collaborative table editing and graphic object grouping are as follows. First, table editing and graphic object grouping are practically useful single-user editing functions and are widely supported in off-the-shelf commercial single-user applications. Supporting collaborative versions of these functions significantly increases the usefulness of collaborative editing systems. Second, among a variety of editing functions, adaptation techniques of table editing and graphic object grouping are

technically representative. The CoTable technique focuses on adapting complex data structures and operations defined on these data structures, and the integration of different object models. CoGroup focuses on resolving application-semantics-level conflicts, achieving desirable effects and adapting complex compound AOs with AO-level mechanisms.

The rest of this chapter is organized as follows. First, the CoTable technique is discussed Section 5.1. Then the CoGroup technique is discussed in Section 5.2. Finally, this chapter concludes with a summary in Section 5.3.

5.1. The TA-Based Collaborative Table Editing Technique

5.1.1. Collaborative Table Editing

Complex information that includes multiple interrelated items is difficult for human beings to comprehend without proper organization. Tables are an efficient way to organize such information. A table is usually defined from two perspectives (Silberhorn 2001). From the presentation-oriented perspective, a table is a two-dimensional structure consisting of rows, columns and cells. From the structure- or content-oriented perspective, a table is a collection of interrelated information items. Each item is semantically associated with multiple categories. Due to these characteristics, tables provide a powerful means for facilitating information organization, comprehension, and comparison (Wang 1996). Because of their usefulness and convenience, tables are supported in a wide range of computer document processing applications such as word processors (e.g. MS Word, OpenOffice Write), web design systems (e.g. MS FrontPage, Macromedia Dreamweaver), and spreadsheet systems (e.g. MS Excel, OpenOffice Calc).

In their ethnographic interviews with users of spreadsheets, which are a special form of tables, Nardi and Miller (1990) noted that most spreadsheets are developed from collaborative work of users with different expertise. Generally,

collaboration is an essential part of table editing. Collaboration may be involved in both table designing and filling processes (Xia et al. 2005a).

5.1.2. The Data Model Adaptation

According to the TA approach, supporting collaborative table editing involves the adaptation of data and operation models of the single-user application's table editing API. In this subsection, the table editing data model adaptation technique will be discussed first.

To adapt table-related object data models exposed by the API to those of the underlying OT technique (i.e. the XOTDM), it is important to have a clear understanding of these table-related data models.

Table Data Models of Single-User Application APIs

When viewed from the *user interface*, a table is a two-dimensional rectangular data structure, consisting of a collection of *rows* and *columns*. Each row or column consists of a sequence of *cells*. A cell may be associated with a row and a column at the same time. A cell may contain some text or graphic objects, which are in a linear sequence. In this *conceptual model* of tables, objects in a table may have various relationships. First, hierarchical relationships exist in the following object pairs: table–column/row, column/row-cell, and cell–cell content. Second, objects in the same collection form a separate linear sequence. For example, each cell has an ordinal index in a row, with which the cell can be accessed from the cell sequence in the row. The ordinal indices range from 0 to $N-1$ where N is the number of cells in the row. Removing or inserting cells affects indices of other cells that have higher indices in the same row, but does not affect indices of cells in other rows.

When viewed from the API of an application, the *table data model* may or may not correspond to the conceptual model. Typically, there are three categories of API table data models, as shown in Figure 5.1.

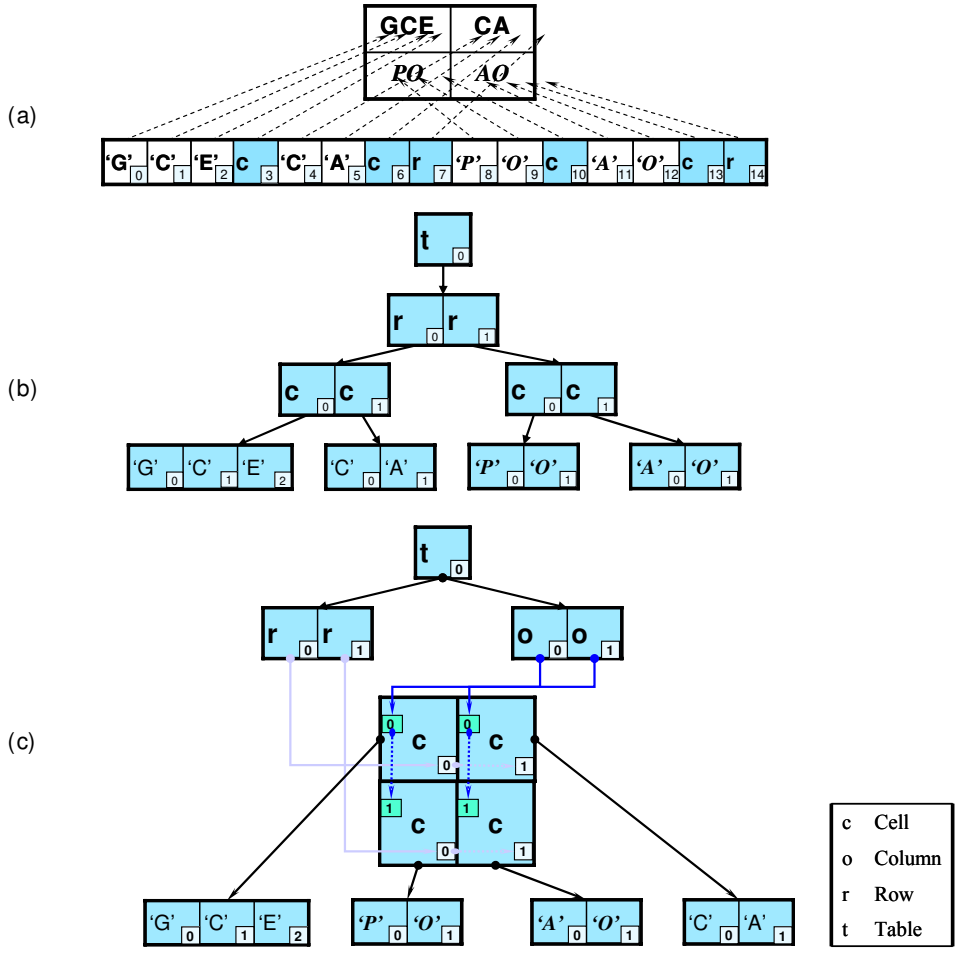


Figure 5.1 Table-related data models in APIs of different single-user applications. (a) The single linear tree data model; (b) the row-based tree data model; (c) the two dimensional data model. The numbers at the lower right corners of each cell stand for object positions in corresponding linear sequences.

(1) *Single linear data model*. In this data model, table data objects can be accessed from a linear addressing space. Inserting or removing objects contained in cells of a table may affect positions of other data objects. Moreover, row/column and cell objects also have marks and occupy positions in the linear addressing space. Therefore, this data model can be represented as a single linear sequence, as shown in Figure 5.1-(a). This data model can be found in APIs of some word-processing applications including MS Word.

(2) *Row-based tree data model*. The most significant feature of this data model is the absence of columns, and table data objects can be accessed from the row-

dimension only. Hierarchical relationships between table-row, row-cell and cell-cell content still exist, and the linear relationships between objects in the same collection also remain. This data model can be represented as a row-based tree, as shown in Figure 5.1-(b). This data model can be found in APIs of some HTML editors, such as MS FrontPage.

(3) *Two-dimensional data model*. In this data model, table data objects can be accessed from both the row-dimension and the column-dimension. This data model directly matches the conceptual model of tables and can be represented as a hierarchical graph, as shown in Figure 5.1-(c). This data model can be found in APIs of a variety of single-user applications, including MS Excel, MS PowerPoint, OpenOffice Writer, and OpenOffice Calc.

Table Data Model Adaptation Schemes

With the variations of API table data models, different adaptation schemes are needed to adapt these data models to the XOTDM. Here the adaptation schemes for the three API data models in Figure 5.1 are discussed respectively.

First of all, the *single linear data model* is a special form of the XOTDM in which only the root level addressing group exists and there is only one linear addressing domain in this group. In the *single linear data model*, an object is uniquely addressed with an integer as the position in the linear sequence. This address is also a special form of the (*n, p*) pair vector address of the OT where there is only one pair in the vector, and the *n* parameter in this pair can be set to a constant since there is only one linear addressing domain.

Moreover, the *row-based tree data model* is also a special form of XOTDM, in which (1) the total number of levels is 4, (2) terminal objects exist only at level 3, and (3) there is only one linear addressing domain in each addressing group. In the *row-based tree data model*, an object is uniquely addressed with a vector of integer. This address is also a special form of the OT vector address where $k \leq 3$

(*k* is the number of (*n, p*) pairs in the address; see Chapter 3), and the *n* parameters in each pair can be set to a constant for the same reason.

Finally, the *two-dimensional data model* is not directly compatible with XOTDM due to the dual hierarchical relationships between cells and rows/columns. However, a comparison of the *two-dimensional data model* and the *row-based tree data model* reveals that removal of column objects from the *two-dimensional data model* reduces the dual hierarchical relationships to a single one and hence converts the *two-dimensional data model* to the *row-based tree data model*.

In summary, the three API data models are all adaptable to XOTDM. The *single linear data model* and the *row-based tree data model* are adapted directly; and the *two-dimensional data model* is adapted after a conversion to the *row-based tree data model*.

Integrating the Table Data Model in Complex Documents

The data model adaptation schemes not only provide a solution to mapping the API table data model into that of OT, but also are the key to integrating tables into the global addressing space of the complex document, as shown in Figure 5.2.

The complex document in Figure 5.2-(a) consists of three linear object sequences: a header, a footnote and a main text. The main text includes three parts. The first line contains an inline graphic object “Hello”, followed by a *return* character. Afterwards there is a table containing two columns and two rows. The footnote mark is in the first cell of the table. The last line contains some text. Suppose the API exposes the same data model for non-table objects as Word. The document is adapted into a tree of linear addressing domains, in which the header objects are in the section addressing domain, and the footnote objects are in the footnote addressing domain (see Chapter 3).

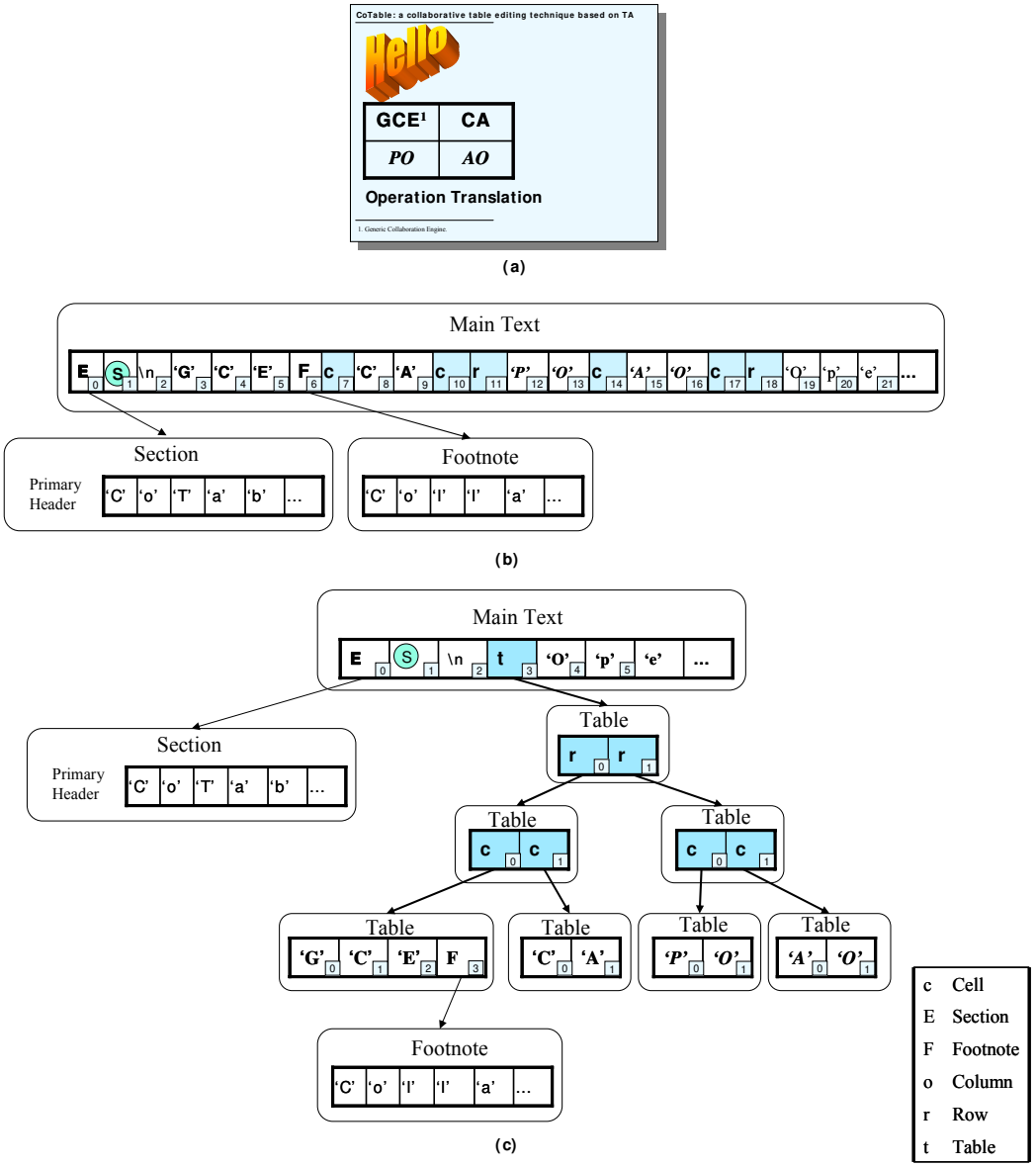


Figure 5.2 Integrating the table into the global addressing space of the complex document. (a) The user's view of the complex document, (b) the data model in which the table is adapted to a single linear sequence, (c) the data model in which the table is adapted to a row-based tree.

In the main text, the graphic object and the text segment can be mapped into two linear sequences separated by the table. Based on different table data models exposed by the API, the table can be adapted to a single linear sequence or a row-based tree. Both adapted data models can be merged with the linear sequence of objects outside the table. The merged data models of both cases are shown in Figure 5.2-(b) and (c), respectively. It is clear that both merged models are

compatible with the XOTDM. All objects in the document can be accessed with a vector of (*n, p*) pairs. For example, the first character in the footnote can be accessed with the vector address [("*Main Text*", 6), ("*Footnote*", 0)] in the *single linear data model* (Figure 5.2-(b)), and can be accessed with the vector address [("*Main Text*", 3), ("*Table*", 0), ("*Table*", 0), ("*Table*", 3), ("*Footnote*", 0)] in the *row-based tree data model* (Figure 5.2-(c)).

Discussion

There are some issues worth discussing in the above data adaptation schemes. First, while adapting the *two-dimensional data model*, it is theoretically equivalent to remove either columns or rows, because both a row-based and a column-based tree can be adapted to XOTDM. Without losing generality, the following discussions will be based on the assumption of a row-based tree.

Second, the row-based tree converted from the *two-dimensional data model* does not need to be semantically equivalent to its original two-dimensional form. The conversion process selectively preserves some information about the table structure but discards the rest, including the hierarchical relationships between cells and columns. This is acceptable because the XOTDM needs to maintain only information relevant to OT. For example, OT needs to know only the vector address of a cell object in the XOTDM, regardless which column the cell is subordinate to, so information about columns can be ignored. However, it is important for OT to know that one cell is located before another in the same collection, so such information is retained.

5.1.3. Table Operation Model Adaptation

Table-related AOs could target objects contained in table cells (e.g. text or graphics) or table structure objects (e.g. cell or row). The data model adaptation schemes have integrated objects in the table into the global data model of the

whole document, so AOs used to manipulate objects (e.g. text or graphics) outside a table can also be used for table content objects, and the operation adaptation techniques for existing AOs can be directly inherited. However, table structure operations are table-specific and cannot be supported by existing AOs designed for graphics or text (see Chapter 3). They require special adaptation techniques. Therefore, the following discussion on CoTable operation model adaptation focuses on the table editing operations only.

The solution to bridging the gap between operation models of the table editing API and OT is to define a set of table structure AOs, denoted as AO_t. As a vehicle for the translation between the API and POs, the AO_t should (1) correctly reflect the table editing API's effects by covering all affecting factors, and (2) facilitate the translation between the table editing API and PO.

Following the operation model adaptation strategy in Chapter 3, AO_t are organized in two dimensions. One dimension the types of table structure object that the AO_t targets. In this dimension, there are three table structure object types: *row*, *column* and *cell*. Therefore, there should be three AO_t categories in this dimension, which are *Row*-AO_t, *Column*-AO_t and *Cell*-AO_t. The other dimension is the PO types. The three AO_t categories in this dimension include *Insert*-AO_t, *Delete*-AO_t, and *Update*-AO_t.

Based on this two-dimensional classification, any AO_t can be placed in a suitable cell in Table 5.1. In fact, there are many more AO_t in real applications than those listed in Table 5.1. For example, additional *Cell-Update*-AO_t may include *Change_CellFillColor*, *Change_CellBorderStyle*, *Change_CellBorderColor*. Nevertheless, for the purpose of investigating issues of operation translation, the AO_t listed in Table 5.1 are representative and adequate.

Parameters of the AO_t show that they are defined directly on the XOTDM. The parameter *vp* is a vector of *(n, p)* pairs. It indicates the starting position of an AO_t effect range in the XOTDM. The parameter *len* indicates the length of an AO_t

effect range. Apart from positional references, other OT-relevant parameters are also kept. For *Insert*- and *Delete*-AO_t, objects affected by the AO_t are kept as the last parameter: *row*, *col* or *cell*. For *Update*-AO_t, the old value *o_val* and new value *n_val* of the target attributes are recorded. These parameters are needed in OT for consistency maintenance and group undo (Sun 2002a).

Table 5.1. AO_t classification.

PO \ *Obj*	**Row**	**Column**	**Cell**
Insert	***Ins_Row***(*vp, len, row*)	***Ins_Col***(*listOf<vp, len>, col*)	***Ins_Cell***(*vp, len, cell*)
Delete	**Del_Row**(*vp, len, row*)	**Del_Col**(*listOf<vp, len>, col*)	***Del_Cell***(*vp, len, cell*)
Update	***Change_rowHeight*** (*vp, len, o_val, n_val*)	***Change_ColWidth*** (*listof<vp, len>, o_val, n_val*)	***Change_cellColor*** (*vp, len, o_val, n_val*)

The effect range parameters (*vp* and *len*) are able to locate any continuous range in XOTDM. Therefore, for an AO_t that has a single continuous effect range (*Row*- or *Cell*-AO_t), the effect range parameters are sufficient in any API data models. However, a *Column*-AO_t has dispersed effect ranges in both *single linear* and *row-based tree data models*, so a list of effect range parameters is needed.

With the AO_t definition in Table 5.1, the translation from the AO_t to both PO and the API are straightforward. In AO_t-PO translation, the PO type is just the PO category of the AO_t; parameters of the PO can be directly taken from the OT-relevant parameters of the AO_t. A *Row*- and *Cell*-AO_t are basic AOs, so they are translated into individual POs. However, *Column*-AO_t are compound AOs, so they should be translated into sequences of POs due to their dispersed effect ranges. Each PO represents the effect on a single cell. On the other hand, while interpreting AO_t with the API, the effect range parameters are used to locate the target object in the API data model; the target object type encoded in the AO_t type provides information about the target object's API interface (e.g. method definitions); the PO type encoded in the AO_t type is used to choose the method to invoke; other AO_t parameters are used as method invocation parameters.

5.1.4. Supporting Collaborative Table Editing in CoWord

The CoTable technique has been implemented in the CoWord and CoPowerPoint systems. Application-specific issues that emerged in adapting data and operation models of Word table-editing API will be discussed in this subsection.

Special Issues in Word Table Data Adaptation

As discussed in Chapter 3, the Word API exposes a data model compatible with XOTDM. In a Word document, the table structure is organized as the *single linear data model*. Objects inside table cells and outside ones (but in the same document element, such as main text, comment or footnote) are mapped into the same linear addressing domain and can be accessed with their positional references, as shown in Figure 5.1-(b). Moreover, there are *end-of-cell* and *end-of-row* marks for each cell and row in the linear addressing domain with unique positions.

However, some objects in a Word document are hidden in both the user interface and the API. To ensure the correctness of the data address adaptation, it is important that these objects also be located and mapped to the XOTDM. One example of such hidden objects is the invisible cells generated while handling irregular tables.

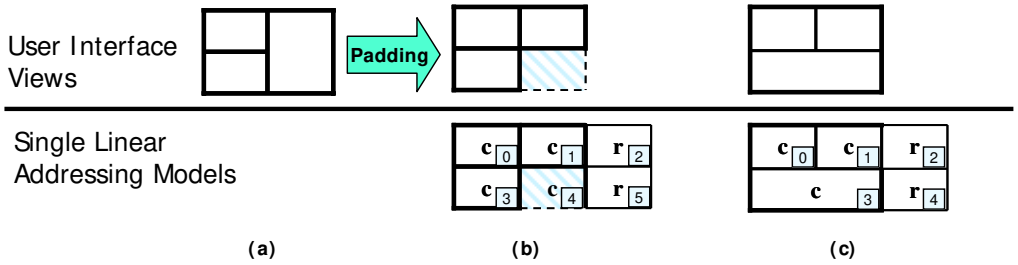


Figure 5.3 Handling irregular tables and its effects on the data model. (a) A row-irregular table; (b) the padding effect on the data model; (c) a column-irregular table. To better match the user interface views of tables, *single linear data models* are shown in rectangular forms.

Some Word tables are irregular, in the sense that some cells cannot be definitely subordinated to certain rows or columns. Figure 5.3[9]-(a) and (c) show tables that are irregular in two different dimensions.

In the row-irregular table in Figure 5.3-(a), an ambiguity exists in determining which row the right cell belongs to, because it spans two rows. In the Word API data model, this cell is associated with the upper row. At the same time, an invisible cell is padded beneath the spanning cell in the lower row to eliminate the ambiguity (shown in Figure 5.3-(b)). In contrast, in the column-irregular table in Figure 5.3-(c), no padding is needed.

Such invisible cells must not be ignored in the data model adaptation. Although these cells are invisible in the user interface and inaccessible from the Word API, they are also assigned with positions in the global linear addressing space. Ignoring these cells would have the consequence of ruining the correctness of the data model adaptation.

Special Issues in Word Table Operation Adaptation

In the Word table operation adaptation, there are also some special issues worth discussing. The first one is the approach to supporting irregular tables.

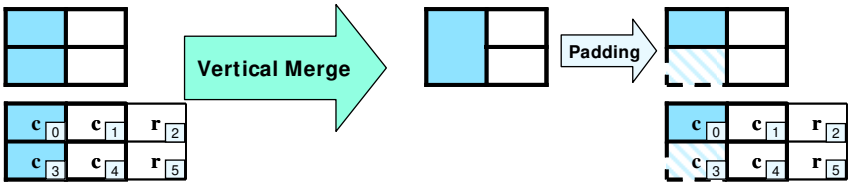


Figure 5.4 Effects of vertical cell merge on the user interface and data model.

As a TA-based system, CoWord generates AO_t by intercepting the user's table-editing interactions with the Word user interface; the user's interactions may trigger Word table-editing functionalities to change the document state. Therefore,

[9] Addresses of table structure objects are vectors of (*n, p*) pairs. In this figure, only their linear indices in the leaf-level linear addressing domain are shown to simplify the discussion.

an important basis for the AO_t generation is a precise understanding of these functionalities' effects.

Word table-editing functionalities have visible effects on the user interface and invisible effects on the API data model. In most cases, these effects are consistent, but sometimes they may be inconsistent. Under any circumstances, the generation of AO_t should always be based on the API data model effects.

One example where this inconsistency occurs is the vertical cell merge, whose effects on the user interface and the data model are shown in Figure 5.4. When two cells are merged vertically, the effects on the user interface is that the lower cell is removed and the upper cell spans two rows. This vertical merge causes irregularity, so the padding scheme is applied (by Word) in the data model. As shown in Figure 5.4, there is no positional difference between the data model states before the merge and after the padding. The only difference is that the lower cell becomes invisible. According to this data model effect, a *Cell-Update*-AO_t needs to be generated to set the *visibility* attribute of the lower cell to false.

Another issue is to preserve regularity effects of AO_t. In the single-user environment, only *Ins_Cell* and *Del_Cell* AO_t could irregularize a regular table; the application of a *Row/Column*-AO_t to a regular table preserves the regularity of the table. This regularity effect of the AO_t should be preserved in the collaborative environment. However, in the *single linear data model* of the Word API and in the face of concurrency, the regularity effect may be lost without special treatment.

As shown in Figure 5.5, from the initial table state (shown in Figure 5.5-(a)), site 1 generates an AO_t O_1 that inserts a new column. Concurrently site 2 generates an AO_t O_2 that inserts a new row. Both *Insert*-AO_t contain two cells. After executed locally (shown in Figure 5.5-(b)), they are propagated to remote sites. When O_1 arrives at site 2, it is translated to POs, processed by OT and executed, which results in the insertion of two cells and leads to the table state

shown in Figure 5.5-(c). Site 1 goes through a similar process after the arrival of O_2 and reaches the same table state.

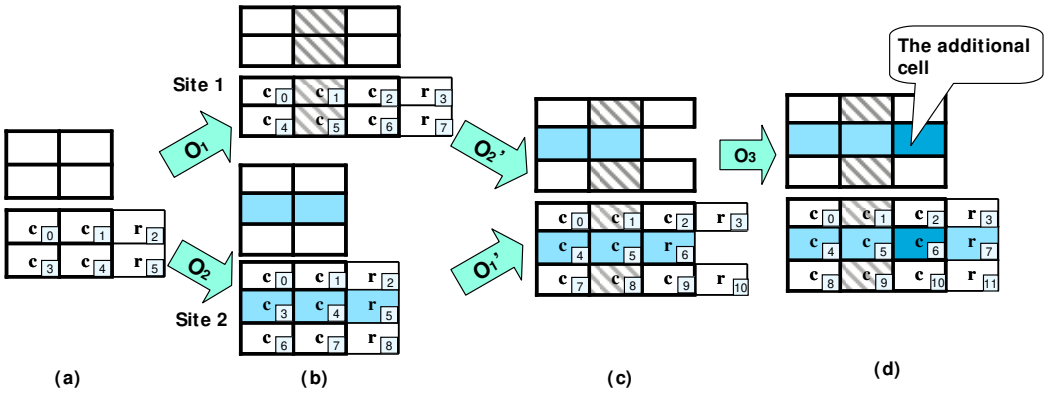


Figure 5.5 Preserving the regularity effects of *Ins_Row* and *Ins_Col* AO_t. (a) The initial state; (b) after local execution; (c) after remote execution; (d) after the execution of the addition AO_t. In this figure, O_1=*Ins_Col* (<1, 1>, <5, 1>, *col*); O_2=*Ins_Row* (3, 3, *row*); O_3=*Ins_Cell* (6, 1, *cell*); O_1' and O_2' are OT-processed forms of O_1 and O_2.

The table in Figure 5.5-(c) is an irregular one, whose irregularity comes from the combined effect of two concurrent *Row*- and *Column*-AO_t. In other words, the regularity effect of these two AO_t is lost in the face of concurrency.

The correct combined result of these two AO_t should be that shown in Figure 5.5-(d), where the regularity is still preserved after the insertion of a row and a column. The difference between the tables in Figure 5.5-(c) and (d) is that the one in Figure 5.5-(d) has an additional cell, which helps preserve the table's regularity. To convert the table state from that shown in Figure 5.5-(c) to (d), an additional *Ins_Cell* operation O_3 is needed to insert that additional cell.

A thorough investigation shows that this problem occurs only when a column AO_t (i.e. *Ins_Col*, *Del_Col* and *Upd_Col*) and a concurrent *Ins_Row* AO_t target the same table. An additional AO_t needs to be generated in these cases to preserve the table's regularity.

5.1.5. Comparison to Other Collaborative Table Editing Techniques

Prior work on collaborative table editing has been restricted to collaboration-aware table-centric (spreadsheet) applications. The CoTable technique is unique in providing a collaborative table editing solution to both table-centric and word-centric applications.

Super Spreadsheet (Fuller et al. 1993) is a collaborative spreadsheet system for face-to-face users. Management of concurrency, spreadsheet version and history is performed in an object-oriented way. For concurrency control, a transaction-based approach has been adopted. The user's interactions with the system are organized as transactions. During the execution of a transaction, implicit locks are used to lock the data objects before updating (i.e. pessimistic locking), and locks are released at the transaction commitment time, which is chosen by the user explicitly. Locks of multiple objects can be acquired in arbitrary orders (i.e. non-strict 2-phase locking), so deadlock is possible. There exist automatic deadlock detection mechanisms in the system but users must be involved in deadlock resolution by negotiation. This solution works well in the local-area network environment (for face-to-face users). However, if this approach were applied in the Internet environment, the system responsiveness may suffer due to the use of pessimistic locks.

Similarly, the Shared Spreadsheet (WARP 2006) also takes a transaction-based approach as its concurrency control mechanism. Users need to explicitly start a transaction before editing and end the transaction afterwards. Transactions failing in conflicts have to be rolled back, which may result in the loss of collaborative work. Besides, a series of auxiliary features has been implemented to increase performance and reduce the possibility of rolling back.

Transaction/lock-based concurrency control solutions are able to protect data integrity by prohibiting conflicting updates on shared data objects, which is

important in achieving *semantic consistency* (Dourish 1996; Sun and Ellis 1998) in collaborative applications. On the other hand, OT-based solutions can ensure *syntactic consistency* (characterized by *convergence, intention-preservation,* and *causality-preservation* (see Chapter 2)), and provide high responsiveness, fine-grain concurrency, and a high degree of freedom to the users in their interactions with the shared application in the Internet environment. OT and transaction/locking are complementary techniques and could be integrated for achieving both syntactic and semantic consistency.

An OT-based distributed collaborative spreadsheet system was proposed by Palmer and Cormack (1998). Their OT technique is specially designed for supporting the *two-dimensional data model*, and spreadsheet-specific operations: *insert* and *delete* rows or columns in a table, and *set*, *format*, and *copy* the cell value of a table. These spreadsheet-specific operations are at the same level as the AO_t in CoTable. In contrast, CoTable is based on an OT technique which directly supports only three generic primitive operations (*Insert, Delete*, and *Update*), and an adaptation technique to map application-level table editing operations (i.e. the AO_t) into these primitive operations. The benefits of the CoTable approach is the reduced complexity in designing transformation functions and the reusability of transformation functions for supporting a wide range of data types in complex documents.

5.2. The Collaborative Graphic Object Grouping Technique

5.2.1. Collaborative Graphic Object Grouping

Documents of graphics editing applications (e.g. slides authoring systems and CAD systems) often contain a large number of objects with complex logical structures. Managing complex structures on the basis of individual objects would cost significant efforts or sometimes may be infeasible. Object grouping, which packs multiple logically related objects into a single *group-object* and vice versa,

is an effective means to help manage the complexity of graphics editing. When objects are grouped, they behave like a single object in response to modifications to any attribute. At the same time, the user can also choose to modify some attributes (e.g. fill color) of group members individually. Furthermore, a group-object can be a member in another group-object, which provides a multi-level hierarchical structure for managing complex documents. In summary, object grouping not only protects the logical relationship among group members against mistaken actions, but also provides the convenience of modifying group members individually (Xia et al. 2005c).

Supporting collaborative object grouping is nontrivial due to the increased complexity in both the data and operation models. First, existing collaborative graphics editing techniques often treat graphic objects as independent entities, but object grouping introduces group relationships among graphic objects. Second, existing collaborative graphics editing techniques focus on supporting three types of *basic* operations: a *CreateObj* operation creates a new object (e.g. a line, circle, square or textbox); a *DeleteObj* operation removes an existing object; and a *ChangeAtt* operation changes an attribute (e.g. size, color or position) of an existing object. Object grouping requires support for two additional operations: a *Group* operation packs a collection of objects into a single group-object; and an *Ungroup* operation unpacks a group-object into a collection of individual objects. In this chapter, the term *grouping operation* is used to mean either a *Group* or an *Ungroup* operation.

5.2.2. Conflict Resolution in the Presence of Grouping Operations

Conflict Relations among Operations

The main technical challenge in supporting collaborative graphic object grouping is conflict resolution and consistency maintenance in the presence of group-objects and grouping operations in a TA-based real-time collaborative

environment. As discussed in Chapter 4, conflicts may occur when multiple users concurrently update the same attribute of a common object. Moreover, two concurrent *Group* operations may also conflict with each other if they target common objects since these common objects cannot belong to two different result group-objects at the same time.

Before designing conflict resolutions, the conflict relation between graphics editing operations is defined as follows. To define the conflict relation, the following notions are used: (1) *Type*(O) denotes the type of operation O; (2) *Target*(O) denotes the set of identifiers of target objects of operation O; and (3) *Att.Key*(O) denotes the attribute type of operation O if O is a *ChangeAtt* operation.

Definition 5.1. Conflict relation "$\otimes$". Two operations O_1 and O_2 conflict with each other, expressed as $O_1 \otimes O_2$, if and only if (1) O_1 and O_2 are concurrent; (2) $Target(O_1) \cap Target(O_2) \neq \Phi$; and (3)

a. $Type(O_1) = Type(O_2) = Group$; or
b. $Type(O_1) = Type(O_2) = ChangeAtt$ and $Att.Key(O_1) = Att.Key(O_2)$.

Definition 5.2. Compatible relation "$\odot$". Two operations O_1 and O_2 are compatible, expressed as $O_1 \odot O_2$, if and only if they do not conflict with each other; that is, $\neg(O_1 \otimes O_2)$.

According to the above definitions, sequential operations are compatible; operations without common target objects are compatible; and operations of different types are compatible. Conflict relations occur only between a pair of *Group* operations or a pair of *ChangeAtt* operations under the conditions specified in *Definition 5.1*. The conflict/compatible relations among the three basic operations and the two grouping operations are summarized in Table 5.2 (called a conflict relation triangle in Sun and Chen (2002)). The meaning of shaded cells will be explained later in this chapter.

Table 5.2. The conflict relation triangle of five operation types.

	CreateObj	*DeleteObj*	*ChangeAtt*	*Group*	*Ungroup*
CreateObj	⊙	⊙	⊙	⊙	⊙
DeleteObj		⊙	⊙	⊙	⊙
ChangeAtt			⊗/⊙	⊙	⊙
Group				⊗/⊙	⊙
Ungroup					⊙

In Chapter 4, the *Multi-Version Single-Display* (MVSD) conflict resolution strategy and its suitability for TA-based collaborative systems have been discussed. This strategy is also adopted in the TA-based CoGroup technique to resolve conflict among *Group* and *ChangeAtt* operations.

Combined Effects for Conflict and Compatible Operations

Based on the conflict/compatible relations given in Table 5.2 and the MVSD strategy, the combined effects among the five operations *CreateObj, DeleteObj, ChangeAtt, Group*, and *Ungroup,* are specified in this subsection.

According to Table 5.2, a *CreateObj* operation is always compatible with all operations, including another *CreateObj* operation, because the object to be created cannot be targeted by another concurrent operation.

A *DeleteObj* operation is always compatible with all other operations as well because the effect of a *DeleteObj* operation can be combined with the effect of any other concurrent operation targeting the same object.

(1) The combined effect with another *DeleteObj* operation is the deletion of the target object (Figure 5.6-(b)). Their effects have been combined in the sense that the deleted object can be recovered only after undoing both operations.
(2) The combined effect with a *ChangeAtt* operation is the change of the attribute and the deletion of the target object (Figure 5.6-(c)).

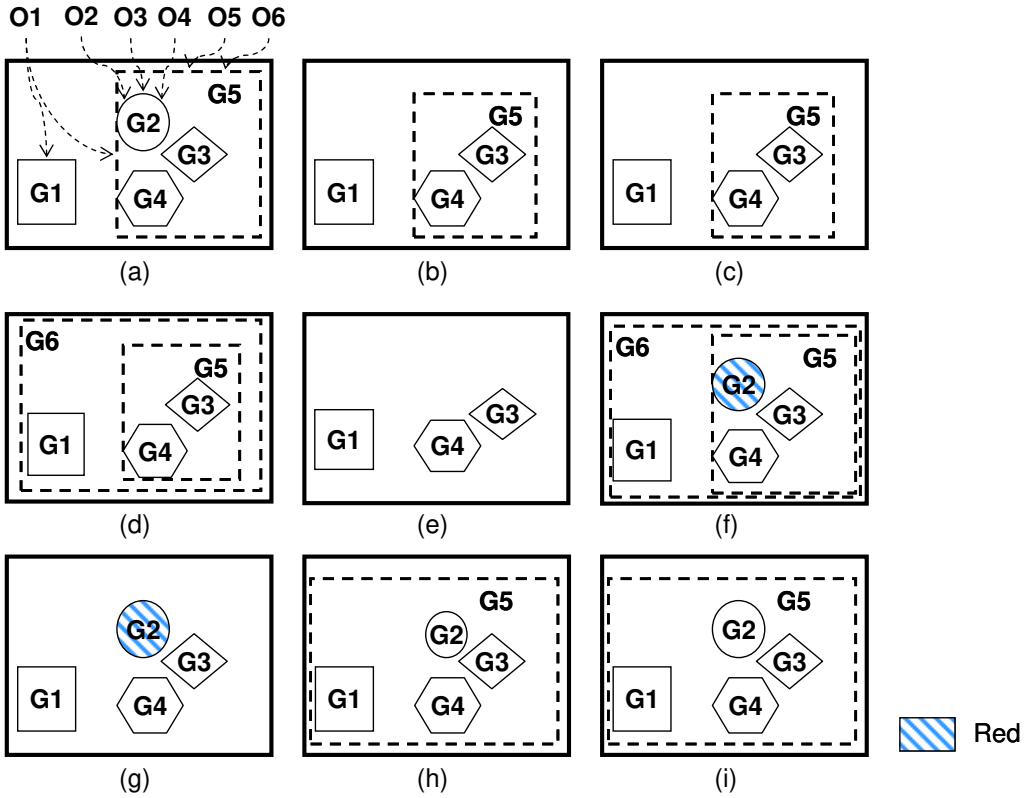


Figure 5.6 Combined effects between graphics editing operations: (a) the initial document state and operations: O1 = *Group*(G1, G5); O2 = O3 = *DeleteObj* (G2); O4 = *ChangeAtt* (G2, FillColor, red); and O5 = O6 = *Ungroup* (G5); and the combined effects between (b) O2 and O3, (c) O2 and O4, (d) O2 and O1, (e) O2 and O5, (f) O4 and O1, (g) O4 and O5, (h) O1 and O5, (i) O5 and O6, respectively.

(3) The combined effect with a *Group* operation is the creation of a group-object containing all member objects targeted by the *Group* operation, except the member object targeted by the *DeleteObj* operation (Figure 5.6-(d)).

(4) The combined effect with an *Ungroup* operation is the unpacking of the member objects in the group-object targeted by the *Ungroup* operation and the deletion of the member object targeted by the *DeleteObj* (Figure 5.6-(e)).

A *ChangeAtt* operation may conflict with another *ChangeAtt* operation under the condition specified in *Definition 5.1*; but it is always compatible with other operations because the effect of a *ChangeAtt* operation can be combined with the effect of any other concurrent operation targeting the same object.

(1) The combined effect with a *DeleteObj* operation is illustrated in Figure 5.6-(c).
(2) The combined effect with a *Group* operation is the creation of a group-object containing all target member objects, and the change of the attribute of one member object targeted by the *ChangeAtt* operation (Figure 5.6-(f))
(3) The combined effect with an *Ungroup* operation is the unpacking of all member objects inside the target group-object, and the change of attribute of the member object targeted by the *ChangeAtt* operation (Figure 5.6-(g)).

A *Group* operation may conflict with another concurrent *Group* operation if they target common objects; but it is always compatible with other operations because the effect of a *Group* operation can be combined with the effect of any other concurrent operation targeting the same object.

(1) The combined effect with a *DeleteObj* or a *ChangeAtt* operation has been illustrated in Figure 5.6-(d) and Figure 5.6-(f), respectively.
(2) The combined effect with an *Ungroup* operation is the creation of a group-object containing all member objects targeted by the *Group* operation and the unpacking of the group-object (a member object targeted by the *Group* operation as well) targeted by the *Ungroup* operation (Figure 5.6-(h)).

An example for illustrating the combined MVSD effects of two conflict *Group* operations is given in Figure 5.7. Initially, the document contains five objects: G_1, G_2, ..., G_5, and suppose two operations O_1 = *Group*(G_1, G_2, G_3) and O_2 = *Group*(G_3, G_4, G_5) are generated concurrently, as shown in Figure 5.7-(a). Since O_1 and O_2 target a common object G_3, they conflict with each other. To achieve the MVSD effect, two versions $G_{3\text{-}O1}$ and $G_{3\text{-}O2}$ should be created to accommodate the effects of both O_1 and O_2, but only $G_{3\text{-}O1}$ is displayed in the group-object created by O_1 (Figure 5.7-(b)), provided that O_1 has a higher priority than O_2. The version $G_{3\text{-}O2}$ is maintained internally in the group-object created by O_2 but is invisible at the user interface due to the single-display strategy. However, after O_1 is undone, $G_{3\text{-}O2}$ will become visible as shown in Figure 5.7-(c).

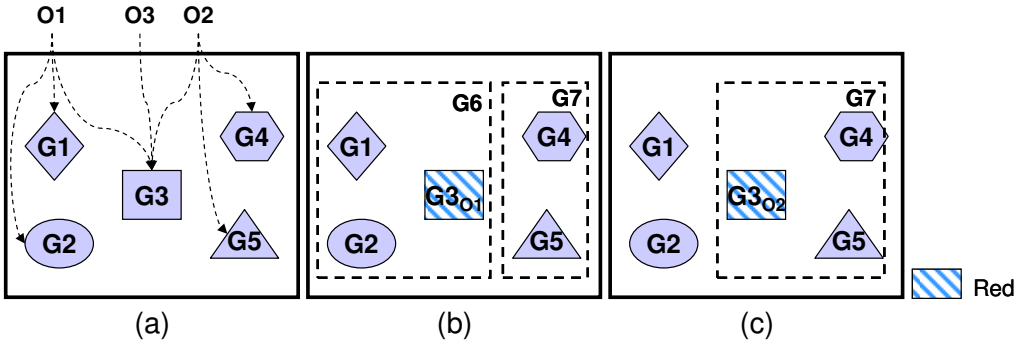


Figure 5.7 An example for illustrating the combined MVSD effect of two conflict *Group* operations.

An *Ungroup* operation is always compatible with other operations for the reasons explained above and illustrated in Figure 5.6-(e), Figure 5.6-(g) and Figure 5.6-(h) respectively. The combined effect of two concurrent *Ungroup* operations targeting the same group-object is the unpacking of the target group-object (Figure 5.6-(i)). Both *Ungroup* operations have been combined in the sense that the group-object can be recovered only after undoing both operations.

5.2.3. The Data Model Adaptation for Graphic Objects

As the first step of supporting collaborative graphic object grouping in the TA framework, the data model adaptation technique of graphic objects, particularly group-objects into a data model that is compatible with that of OT (namely the XOTDM in Chapter 3), will be discussed in this subsection.

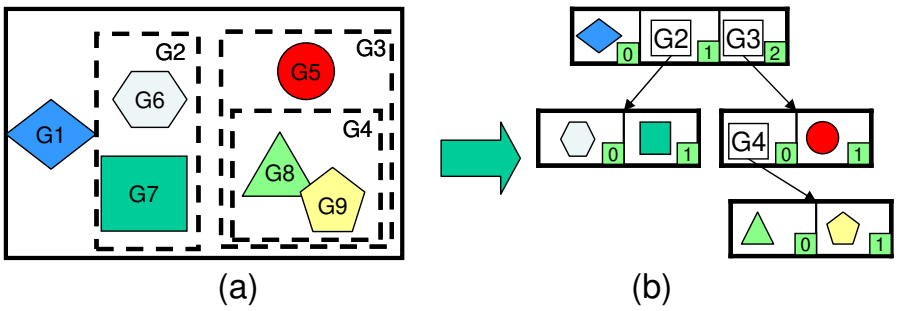


Figure 5.8 The group objects data model. (a) The user interface representation; (b) The data model in the API.

A wide range of graphics editing applications (including Word and PowerPoint) have provided varieties of mechanisms (in their APIs) for mapping any graphic objects, including group-objects, into a tree of linear addressing domains. To illustrate this address mapping, consider the following example: Figure 5.8-(a) shows a graphic document when viewed from the user interface; and Figure 5.8-(b) shows the mapping of the graphic objects in this document to a tree of linear addressing domains when viewed from the API. In this example, the top three objects (G_1, G_2, and G_3) are mapped into the top-level linear addressing domain in the tree; the member objects in the two group-objects G_2 and G_3 are mapped into two second-level addressing domains, respectively; and the member objects in group-object G_4 are further mapped into a third-level addressing domain. As shown in this example, member objects of a group-object form a separate linear addressing domain; a group-object (e.g. G_4) can be a member object of a higher level group-object (e.g. G_2), allowing multiple levels of object grouping.

Under the data model in Figure 5.8-(b), any graphic object can be accessed with the vector address. For example, the address of the pentagon can be expressed as a vector address [2, 0, 1], where "2" refers to the group-object G_3, "0" refers to the group-object G_4; and "1" refers to the pentagon object. Comparing with the XOTDM (see Chapter 3), the data model in Figure 5.8-(b) is a special case of XOTDM in which there is only one linear addressing domain in every addressing group, like the *row-based tree data model* in the CoTable technique (see Section 5.1.2). The vector of integer address for accessing graphic objects can be easily converted into the vector of *(n, p)* pairs address in the XOTDM. While the CoGroup technique is applied in applications in which there are multiple linear addressing domains in each node (i.e. addressing group) like Word and PowerPoint, the domain identifier can be attached to the top level integer and constants can be attached to integers at lower levels, so that a vector of integers is converted into a vector of (*n, p*) pairs. For example, the above integer vector [2, 0, 1] is converted into [("Main Text", 2), ("Graphic Group", 0), ("Graphic Group", 1)] in CoWord if these graphic objects exist in the main

document. To simplify the discussion, the integer vector is used in the rest of this chapter to address graphic objects.

5.2.4. The Operation Model Adaptation for Group Operations

The second step of supporting collaborative graphic object grouping in the TA framework is to adapting graphics editing functions to the operation model of OT, which contains three POs: *Insert*, *Delete* and *Update* (see Chapter 4).

Basic AOs targeting Group-Objects

Our strategy for adapting graphics editing functions is to define a set of graphics editing AOs, called AO_g. For the three basic graphics editing operations, there are three corresponding basic AO_g: *CreateObjAO*$_g$, *DeleteObjAO*$_g$, and *ChangeAttAO*$_g$. Effects of these basic AO_g in the group-object data model can be fully captured by POs, so the built-in mechanisms of OT are capable of resolving conflicts among basic AO_g without any additional mechanisms at the AO level.

An example of resolving conflicts among *ChangeAttAO*$_g$ targeting group-objects is shown in Figure 5.9. From the initial document state (Figure 5.9-(a)), three operations are generated concurrently: O_1 = *ChangeAttAO*$_g$([0, 0, 0], *FillColor*, *Red*) to change the filling color of non-group object G_1 into *Red*, O_2 = *ChangeAttAO*$_g$([0, 0], *FillColor*, *Green*) to change the filling color of group-object G_5 to *Green*, and O_3 = *ChangeAttAO*$_g$([0], *FillColor*, *Blue*) to change group-object G_6 to *Blue*. According to the conflict definition (*Definition* 5.1), these three AO_g conflict. Assume their priority relation is $O_1 > O_2 > O_3$.

The conflicts among these AO_g can be detected in OT from their common PO types (all are type *Update)*, the same target attribute type (all are *FillColor*), and overlapping target ranges, (O_3*.addr* is the prefix of O_2/O_1*.addr*, and *O*2*.addr* is the prefix of O_1*.addr*). These conflicts can be solved with the conflict resolution algorithm for the *Update* PO (Sun et al. 2004) and the combined MVSD effects

shown in Figure 5.9-(b) are achieved. In this result, multiple versions for objects targeted by conflict AOs are created, but only the versions created by AO_g with the highest priorities (e.g. $G_{1\text{-}O1}$, $G_{2\text{-}O2}$ and $G_{3\text{-}O2}$) are displayed.

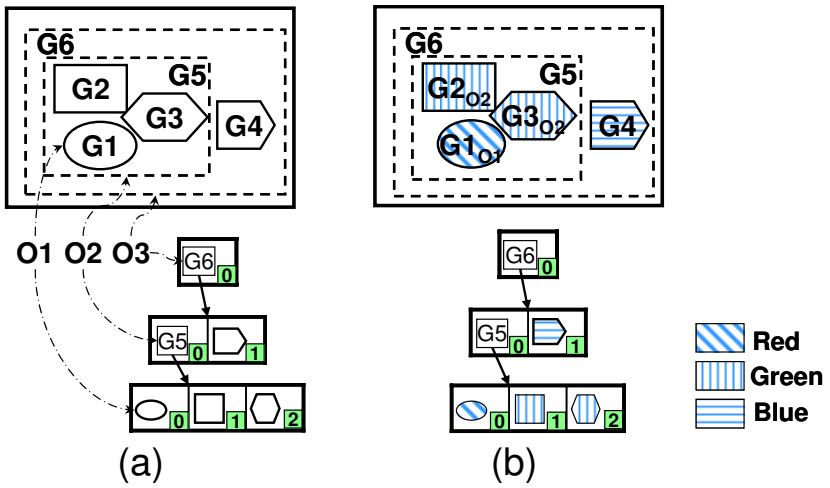


Figure 5.9 A scenario of three conflict *ChangeAttAO$_g$*.

Grouping AO_g Representation

For object grouping, there are two grouping AO_g, named as *GroupAO$_g$* and *UngroupAO$_g$*, respectively. To determine the representation of these grouping AO_g, it is necessary to analyse their effects on both the real objects (visible from the user interface) and the object data model (visible from the API).

As illustrated in Figure 5.10, the effect of a *GroupAO$_g$* on the real objects is to pack multiple target objects into a single group-object; and its effects on the internal addressing model include: (1) inserting a group-object in the current addressing domain (at the position before the first target object); and (2) moving all target objects into a lower level addressing domain (linked to the group-object). In moving these target objects, their original relative sequence relationships are preserved (see Figure 5.10-(b)).

The effect of an *UngroupAO$_g$* on the real objects is to unpack the target group-object into multiple member objects; and its effects on the data model include: (1)

moving all member objects to the position of the target group-object in the higher level addressing domain; and (2) deleting the target group-object (see Figure 5.10-(c)).

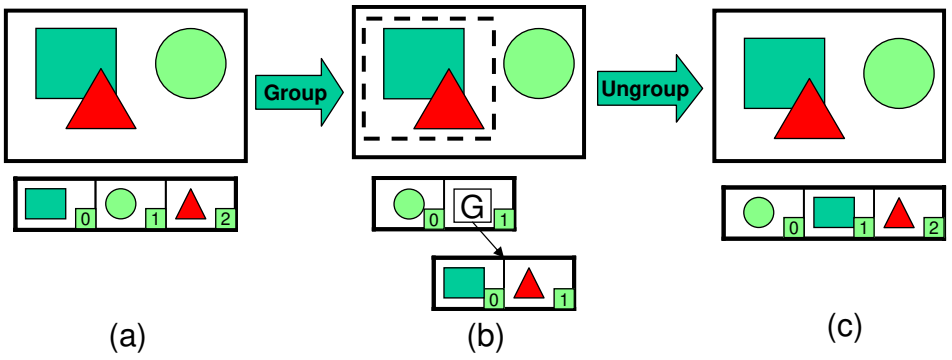


Figure 5.10 Effects of *GroupAO*$_g$ and *UngroupAO*$_g$. (a) The initial state; (b) the state after grouping; (c) the state after ungrouping.

It should be pointed out that after executing the *UngroupAO*$_g$ operation, the document state returns to the previous state before the execution of the *GroupAO*$_g$ operation at the user interface; but the internal addresses of these objects are not restored, as can be seen by comparing Figure 5.10-(a) and (c). These object grouping effects are supported by the APIs of a wide range of single-users applications, including MS Word, MS PowerPoint and OpenOffice Presentation.

To facilitate grouping AO$_g$ adaptation, their representations must capture their effects on both the data objects (needed for replaying their effects in *AO-API Adaptation* (see Chapter 3)), and on the object addressing space (needed for OT-processing in *AO-PO Adaptation)*. Since both *GroupAO*$_g$ and *UngroupAO*$_g$ have the effect of moving existing objects between different addressing domains, a new operation, named *MoveAO*$_g$, needs to be introduced, to represent this effect. The *MoveAO*$_g$ can be represented as follows:

- *MoveAO*$_g$ *(from, to, obj)* denotes the effects of deleting the object *obj* at the address *from* and inserting the same *obj* at the address *to*.

Based on the basic AO_g and *MoveAO*$_g$, the two grouping AO_g can be represented as follows:

(1) *GroupAO*$_g$*(CreateObjAO*$_g$*(addr, go), MoveAO*$_g$*(from-1, to-1, obj-1)*, …, *MoveAO*$_g$*(from-n, to-n, obj-n))* denotes the effects of creating a group-object *go* at address *addr* and moving the target member objects *obj-1, ..., obj-n* from addresses *from-1, ..., from-n*, to new addresses *to-1, ..., to-n* at a lower level addressing domain.

(2) *UngroupAO*$_g$*(DeleteObjAO*$_g$*(addr, go), MoveAO*$_g$*(from-1, to-1, obj-1)*, …, *MoveAO*$_g$*(from-n, to-n, obj-n))* denotes the effects of deleting the target group-object *go* at address *addr* and moving the member objects *obj-1, ..., obj-n* from addresses *from-1, ..., from-n*, to new addresses *to-1, ..., to-n* at a higher level addressing domain.

It should be stressed that the object addresses used in all AO_s are positional references in the data model (see Figure 5.8), rather than the visual locations of the data objects at the user interface.

Grouping AO_g Translation

For processing AO_g with OT, AO_g should be translated into POs. Translation of the basic AO_g is straightforward: a *CreateObjAO*$_g$ has the effect of inserting an object in the data model, so it can be translated into an *Insert* PO; a *DeleteObjAO*$_g$ has the effect of deleting an object from the data model, so it can be translated into a *Delete* PO; a *ChangeAttAO*$_g$ has the effect of changing an attribute of an object in the data model, so it can be translated into an *Update* PO.

On the other hand, *GroupAO*$_g$ and *UngroupAO*$_g$ are compound AO_g in the sense that they cannot be translated into single POs. The translation of a compound AO_g consists of translating each composing AO_g into a list of POs.

Definition 5.3. Translation Rules for Grouping AO$_g$. For each composing AO_g in a grouping AO_g, it is translated as follows:

(1) if the composing AO_g is a basic AO_g: *CreateObjAO_g/ DeleteObjAO_g/ChangeAttAO_g,* then it is translated into a single *PO: Insert/Delete/Update;*
(2) if the composing AO_g is *MoveAO_g*, then it is translated into a pair of POs: *Delete* and *Insert*, where the two POs must refer to the same object (which is different from a pair of independent *Delete* and *Insert*).

Let *GroupAO_g-POList* denote the translated PO list for *GroupAO_g*, *UnGroupAO_g-POList* denote the translated PO list for *UngroupAO_g*. Based on the translation rules in *Definition 5.3*, grouping AO_g are translated as follows:

(1) *GroupAO_g-POList = [Insert(go-addr, go-ref), Delete(from-1, moref-1), Insert(to-1, moref-1), ..., Delete(from-n, moref-n), Insert(to-n, moref-n))*.
(2) *UngroupAO_g-POList=[Delete(go-addr, go-ref), Delete(from-1, moref-1), Insert(to-1, moref-1), ..., Delete(from-n, moref-n), Insert(to-n, moref-n))*.

It should be stressed that the translated PO list captures only part of the grouping AO_g effects (including the timestamps for detecting concurrency (Sun et al. 1998) and priorities) that are needed for generic OT processing. Additional application-specific mechanisms are needed to detect and resolve operation conflict at the AO level, as discussed in the following subsections.

Grouping AO_g Conflict Detection

Based on the grouping AO_g representation and translation schemes, conflicts among basic AO_g can be fully detected and resolved by the mechanisms built in the OT technique. However, detection of conflicts among *GroupAO_g* requires the knowledge of operation type *Group* (see *Definition 5.1*), which is unknown to OT. Therefore, conflict detection in the presence of grouping AO_g requires additional mechanisms at the AO level.

According to *Definition 5.1*, a pair of *GroupAO_g* may conflict under three conditions: (1) they are concurrent; (2) they have overlapping target objects; and

(3) they have the same operation type $GroupAO_g$. OT is able to detect the first two conditions by examining the POs translated from $GroupAO_g$, but the third condition must be checked at the AO level. To facilitate the check of the third condition and to propagate the concurrency and overlapping conditions resulting from the PO level to the AO level, bi-directional references are established between each AO and its translated POs. A routine *GetAO(PO)* is provided to get the AO associated with the PO. Moreover, the underlying OT functions have been extended as follows: when a PO_1 is transformed against a concurrent PO_2 and found to have overlapping target objects with PO_2, this finding and PO_2's reference to its associated AO must be recorded in the transformed PO_1. At the AO level, a routine *POConcurrentAndOverlapping(PO_1)* is provided to check whether PO_1 has been found concurrent and overlapping with another operation, and another routine *GetCOAO(PO_1)* is provided to get the AO associated with PO_2. Based on the above extensions, conflict relationship between two $GroupAO_g$ can be determined by invoking the *AOgConflictDetection*( ) routine defined in Figure 5.11.

```
AOgConflictDetection(TPO)
{
   if(POConcurrentAndOverlapping (TPO) == true)
   {
      if(GetAO(TPO).type == GetCOAO(TPO).type == Group)
         return true;
   }
   return false;
}
```

Figure 5.11 The routines for detecting grouping AO conflicts.

Resolving Conflicts among Grouping Operations

OT is able to resolve conflicts among basic AO_g, but additional mechanisms at the AO level are needed to resolve conflicts among $GroupAO_g$. This is because resolving $GroupAO_g$ conflicts requires semantic knowledge of the $GroupAO_g$ and its representation, which are not captured by individual POs and hence are unknown to OT. For the same reason, to achieve combined effects among

compatible AOs in the presence of grouping AOs, additional mechanisms at the AO level are also needed. In other words, resolving conflicts among conflict operations and achieving the combined effects among compatible operations require interaction and collaboration between the underlying OT technique and the *AO-PO adaptation* in the TA framework.

An overall picture of the responsibility distribution between these two components is shown in Table 5.2: the non-shaded cells indicate the sole responsibility areas of the generic OT technique for resolving conflicts and achieving the defined combined effects among basic AO_g; the shaded cells correspond to joint responsibility areas of OT plus additional AO-level mechanisms (in the *AO-PO adaptation*) for resolving conflict and achieving combined effects in the presence of grouping AO_g.

In the following discussion, the following auxiliary functions will be used: (1) *GetMove*(PO_x) returns the composing $MoveAO_g$ from which the PO PO_x is translated; and (2) *GetCOMove*(PO_x) returns the composing $MoveAO_g$ of the grouping AO_g whose reference is recorded in the PO PO_x. Implementation of these functions is straightforward, based on the AO-PO association and AO reference recorded in a transformed PO. Furthermore, the term *Common Target* $MoveAO_g$ (CT-MoveAO) is used to mean a composing $MoveAO_g$ of a grouping AO_g that moves a common target object targeted by another concurrent AO_g.

According to the MVSD combined effect, the conflict between two $GroupAO_g$ is resolved based on their priorities. Given two conflict $GroupAO_g$: O_1 with a higher priority and O_2 with a lower priority, their common target objects should be packed in the group-object created by O_1 and excluded from the group-object created by O_2.

In the $GroupAO_g$ representation, the effects of moving target objects are represented by composing $MoveAO_g$. Therefore, for a pair of conflicting $GroupAO_g$ O_1 and O_2, there must be a *CT-MoveAO* in each of them, which targets

a common target object. Based on this observation, the strategy of resolving the conflict between O_1 and O_2 is as follows:

(1) if the O_1 *is* executed after O_2, the *from* parameter of the *CT-MoveAO* of O_1 should be set to the *to* parameter of the *CT-MoveAO* of O_2, so that the common target object will be moved to the group-object created by O_1.
(2) if O_2 is *executed* after O_1, the *CT-MoveAO* of O_2 should be cancelled so that the common target object is excluded from the group-object created by O_2.

Based on the above strategy, the routine *AOgConflictResolution(TPO)* is defined (Figure 5.12) for resolving the conflict between the $GroupAO_g$ (obtained by calling *GetAO*) from which the *TPO* was translated and the $GroupAO_g$ (obtained by calling *GetCOAO*) with which *TPO* was associated due to their concurrent and overlapping relationship.

```
AOgConflictResolution(TPO)
{
    if(GetAO(TPO).priority > GetCOAO(TPO).priority)
        GetMove(TPO).from = GetCOMove(TPO).to;
    else
        GetMove(TPO).cancelled = true;
}
```

Figure 5.12 The routine for resolving conflicts among GroupAO$_g$.

Based on the MVSD effect, this conflict resolution approach also supports selectively displaying versions that are hidden by default. Assume that between the two conflict $GroupAO_g$ O_1 and O_2, O_1 has a higher priority than O_2. According to the MVSD effect, two versions of the common target object are created, but only the version created by O_1 is displayed. To display the version created by O_2, a simple strategy is to undo O_1. The disadvantage of this strategy is that all O_1's object-packing effects are unnecessarily discarded, including those non-common target objects. To preserve O_1's effects to the maximum extent, a better strategy is to partially undo the composing *CT-MoveAO* of O_1. From the adjustment to this $MoveAO_g$ to resolve the conflict between O_1 and O_2, it is clear

that the effect of this undo is only to move the common target object from O_1's group-object into O_2's, while all other member objects in O_1's group-object are intact.

Achieving Combined Effects for Compatible Operations in the Presence of *GroupAO$_g$*

According to the combined effects of concurrent and compatible operations (see Figure 5.6), their effects should be accommodated on the common target object at the same time.

Here scenarios in which two concurrent and overlapping compatible AO_g are involved and at least one of them is a grouping AO_g will be discussed. Given a pair of AO_g, O_1 and O_2 involved in such a scenario, suppose O_1 is executed after O_2. When O_1 is executed, its parameters need to be adjusted according to the changes caused by O_2 to achieve the combined effect. Next, adjustment strategies for different AO_g type combinations will be discussed.

In the routines discussed in this subsection, the input parameter *TPO* is the transformed PO of the currently processed AO_g (i.e. O_1). With *TPO*, O_1 can be obtained by calling *GetAO; O_2* can be obtained by calling *GetCOAO*; the *CT-MoveAO$_g$* of O_1 can be obtained by calling *GetMove* if O_1 is a grouping AO_g; and the *CT-MoveAO* of O_2 can be obtained by calling *GetCOMove* if O_2 is a grouping AO_g.

Consider the scenario in which O_1 is a *GroupAO$_g$* and O_2 is a *DeleteObjAO$_g$* (see Figure 5.6-(d)). When O_1 is executed, the common target object has been deleted by O_2. Therefore, this object should be excluded from the group-object created by O_1. The *GroupAO$_g$* representation shows that the effect of moving the common target object is represented by the *CT-MoveAO* of O_1, so our strategy for this scenario is to cancel the *CT-MoveAO* of O_1. This strategy also applies to the

AO_g combinations of $UngroupAO_g$ versus $DeleteObjAO_g$[10] (the $DeleteObjAO_g$ targets a member object of the $UngroupAO_g$'s target group-object) and $UngroupAO_g$ versus $UngroupAO_g$.

On the other hand, if O_1 is a $DeleteObjAO_g$ and O_2 is a $GroupAO_g$, when O_1 is executed, its target object has been moved into the group-object created by O_1. The $GroupAO_g$ representation also shows that the current address of the common target object is indicated by the *to* parameter of O_2*'s CT-MoveAO*, so our strategy for this scenario is to set O_1*'s* address to the *to* parameter of O_2*'s CT-MoveAO*. This strategy also applies to AO_g combinations $ChangeAttAO_g/DeleteObjAO_g$ versus $UngroupAO_g$ (the $ChangeAttAO_g/DeleteObjAO_g$ targets a member object of the $UngroupAO_g$'s target group-object), $DeleteObjAO_g$ versus $GroupAO_g$, and $UngroupAO_g$ versus $GroupAO_g$.

Based on the above strategies, the routine for achieving combined effects for concurrent and overlapping $GroupAO_g$ and $DeleteObjAO_g$ is shown in Figure 5.13.

```
CE_GroupDeleteObj(TPO)
{
    if(GetAO(TPO).type == GroupAO)
        GetMove(TPO).cancelled = true;
    else
        GetAO(TPO).addr = GetCOMove(TPO).to;
}
```

Figure 5.13 The routine for achieving combined effects for $GroupAO_g$ and $DeleteObjAO_g$.

Consider the scenario in which O_1 is a $ChangeAttAO_g$, O_2 is an $UngroupAO_g$ and they both target the same group-object. When O_1 is executed, the common target group-object has been unpacked into a continuous range of multiple objects by O_2 (see Figure 5.6-(c)). The $UngroupAO_g$ representation shows that the address and length of the unpacked object range are indicated by O_2's composing *MoveAOs*. Therefore, our strategy for this scenario is to set O_1's effect range (i.e.

[10] In this pair, the former AO_g is the AO_g currently being processed (i.e. O_1), and the latter AO_g is the one concurrent and overlapping with the former (i.e. O_2).

address and length) to cover all unpacked objects. This strategy also applies to AO_g combinations *DeleteObj*AO_g versus *Ungroup*AO_g (the *DeleteObj*AO_g targets the same group-object as the *Ungroup*AO_g) and *Group*AO_g versus *Ungroup*AO_g.

In the scenario in which O_1 is an *Ungroup*AO_g and O_2 is a *ChangeAtt*AO_g, when O_1 is executed, O_2 has applied its effect on all member objects of the target group-object. To make sure that after ungrouping, all the unpacked objects will still have O_2's effect, our strategy is to apply O_2's effect to data objects of all O_1's composing *Move*AO_g.

Based on the above strategies, the routine for achieving combined effects for concurrent *Ungroup*AO_g and *ChangeAtt*AO_g targeting the same group-object is shown in Figure 5.14.

```
CE_UngroupChangeAtt(TPO)
{
    if(GetAO(TPO).type == ChangeAttAO)
        SetEffectRange(GetAO(TPO), GetCOAO(TPO));
    else
    {
        for(i = 0; i < GetAO(TPO).MoveAOList.count; i ++)
            ApplyChangeAtt(GetAO(TPO).MoveAOList[i].obj, GetCOAO(TPO));
    }
}
```

Figure 5.14 The routine for achieving combined effects for Ung*roupAO*$_g$ and *ChangeAttAO*$_g$ (targeting the group-object).

Grouping AO-PO Adaptation Algorithm

With the routines discussed above, the *AO-PO adaptation* in the TA framework can be extended to support grouping AO_g, as shown in Figure 5.15.

First, the input AO_g is translated into a series of POs saved in a PO list. Then, each PO in the list is processed as follows. The PO is first transformed in OT. Then, if this AO_g involves in a *Group*AO_g conflict, the conflict resolution routine is invoked. Otherwise, if this AO_g is overlapping with another concurrent compatible AO_g and at least one of them is a grouping AO_g, the

CompatibleAOgCombinedEffects routine is invoked to apply AO_g level mechanisms for achieving combined effects for compatible AO_g. In the *CompatibleAOgCombinedEffects* routine, suitable routines discussed above are invoked according to AO_g type combinations.

```
AOg-POAdaptation(AO)
{
    POList = TranslateAO(AO);
    for(i = 0; i <POList.count; i++)
    {
        TransformPO(POList[i]);
        if(AOgConflictDetection(POList[i]) == true)
            AOgConflictResolution(POList[i]);
        else if (POConcurrentAndOverlapping(POList[i]) == true &&
                IncludingGroupingAO(GetAO(POList[i]), GetCOAO(POList[i])) == true)
            CompatibleAOgCombinedEffects(POList[i]);
    }
}
```

Figure 5.15. The routines for AO-PO adaptation in the presence of grouping AOs.

5.2.5. Comparison to Other Collaborative Graphic Object Grouping Technique

To the best of our knowledge, the operation serialization technique reported in Ignat and Norrie (2004) is the only prior work on collaborative object grouping in graphic editing systems. Both the *CoGroup* work in this chapter and the work in Ignat and Norrie (2004) address similar issues involved in conflict resolution for a similar collection of graphics editing operations, but these two works are very different in their approaches to conflict definitions, combined effects among conflicting/compatible operations, and techniques for conflict resolution.

The notion of conflict in *CoGroup* is based on the conditions that operations are concurrent, target common objects, and cannot be accommodated in the common target objects. Under this conflict definition, conflict may occur only between *ChangeAtt* operations or between *Group* operations and the relations among all other operations are compatible (as shown in Table 5.2). Operation

conflicts are resolved by an *all-operations-effect* technique: multiple versions of the common target objects are created to preserve the effects of all operations, but one version at a time is displayed at the user interface (the *MVSD* technique). *CoGroup* is based on and extends OT for conflict resolution and consistency maintenance.

The notion of conflict in Ignat and Norrie (2004) is based on the conditions that operations are concurrent and do not commute. Under this conflict definition, conflict may occur not only between *ChangeAtt* operations and between *Group* operations, as in the *CoGroup* technique (see Table 5.2), but also among other operations, as shown Table 5.3 (in which the *ChangeAtt* operation represents the *setColor, SetBckColour, setZ, SetText, translate, scale* operations in Ignat and Norrie (2004)).

Table 5.3. Conflict relation triangle of five operation types in Ignat and Norrie (2004).

	CreateObj	***DeleteObj***	***ChangeAtt***	***Group***	***Ungroup***
CreateObj	⊙	⊙	⊙	⊙	⊙
DeleteObj		⊗/⊙	⊗/⊙	⊗/⊙	⊗/⊙
ChangeAtt			⊗/⊙	⊗/⊙	⊗/⊙
Group				⊗/⊙	⊗/⊙
Ungroup					⊗/⊙

For the purpose of resolving operation conflict, two types of conflict are further distinguished in Ignat and Norrie (2004): *real* conflicts are those which can be resolved by preserving the effect of one of the conflict operations (or none of them); and *resolvable conflicts* are those which can be resolved by combining partial effects of conflict operations. Regardless whether the conflict is real or resolvable, conflict resolution is based on *operation serialization*, which achieves the defined effects either by using operation-specific ordering rules for resolvable conflicts, or by using any priority scheme for real conflicts. Serialization is essentially a *single-operation-effect* or *null-effect* conflict resolution technique (Sun and Chen 2002).

It is well known that the combined effects achievable by an *all-operations-effect* technique cover all combined effects achievable by a *single-operation-effect* technique, but the inverse is not true (Sun and Chen 2002). Furthermore, some combined effects among conflict *Group* operations achievable by *CoGroup* are not achievable by the serialization work in Ignat and Norrie (2004). For example, when two concurrent *Group* operations target some common and non-common objects, they are regarded as conflict operations in both *CoGroup* and the approach in Ignat and Norrie (2004) (a *real conflict*). The combined effects in *CoGroup* are the following: both *Group* operations will succeed in creating their result group-objects; both group-objects contain their non-common target objects, but only one of them has the common target objects displayed (see Figure 5.7). However, the combined effects in Ignat and Norrie (2004) are the following: one of the two *Group* operations will win and create the group-object containing all target objects, but the other one will lose completely and have no any effect (not even the effect of grouping the non-common target objects).

In Ignat and Norrie (2004), achieving the partially combined effects for some resolvable conflicts is the main motivation for disqualifying OT from being applied for this purpose and for devising the new operation serialization technique. As shown in the example in Figure 5.9, however, the partially combined effect in Ignat and Norrie (2004) can be achieved by using the generic OT technique without additional application-level support, and more comprehensive MVSD combined effects can be achieved by extending OT with the application-level adaptation. A major problem with operation serialization is its undoing and redoing conflict operations when they are executed out of the correct conflict resolution order, which may cause potential interface disruption (when the undo/redo effects are visible at the user interface) and major performance overheads. It should be pointed out that the undo/redo involved in operation serialization is different from the collaborative undo capability in OT: the former is initiated by the internal system out of the necessity for resolving conflict among

grouping operations, but the latter is initiated by the external user for the purpose of eliminating the effect of error grouping operations (Sun 2002a).

It is worth pointing out that there exist other alternative approaches to conflict resolution based on locking (e.g. Ensemble (Newman-Wolfe et al. 1992) and GroupDraw (Greenberg and Marwood. 1992)) or different kinds of serialization (e.g. GroupDesign (Karsenty et al. 1993) and LICRA (Kanawati 1997)), but none of them addressed the issues related to collaborative object grouping. The reader is referred to Sun and Chen (2002) for detailed comparisons between the multi-versioning approach, on which *CoGroup* is based, and these alternative approaches.

5.3. Summary

In this chapter, two TA-based advanced adaptation techniques have been discussed. The first one is a collaborative table editing technique, called CoTable, and the second one is a collaborative graphic object grouping technique, called CoGroup.

The CoTable technique includes techniques for adapting data and operation models of table editing APIs. Single-user application APIs provide a variety of data models for accessing table objects. Typical ones are *single linear data model*, *row-based tree data model* and *two-dimensional data model*. The *single linear data model* and the *row-based tree data model* can be directly adapted to that of OT; and the *two-dimensional data model* can be adapted after being converted to the *row-based tree data model*. These three data model adaptation schemes not only map the API table data models to that of OT, but also help integrate tables into a global addressing space of the complex document.

The CoTable operation adaptation technique is to define a set of table structure editing AOs, called AO_t. AO_t are named and grouped in two dimensions: the PO types and the target object types. Translation of AO_t is straightforward. *Row*- and

Cell-AO$_t$ are translated into individual POs, and *Column-AO$_t$* are translated into multiple POs of the same type, because they are compound AOs.

The CoGroup technique involves adapting the data and operation models of graphic editing APIs in the face of object grouping, and resolves conflicts between *Group* operations to achieve the MVSD effect. To resolve the conflict between *Group* operations, the conflict relation and combined effect for conflict and compatible graphics editing operations are defined.

Single-user graphics editing application APIs have provided mechanisms for mapping graphic objects into a tree of linear addressing domains, which meets the data model adaptation requirement. To map graphics editing operations to the OT data model, a set of graphics editing AO, called AO_g are defined, which include basic AO_g and grouping AO_g. Conflicts among basic AO_g can be resolved with built-in mechanisms of OT.

Both detecting and resolving conflicts among *Group* AO_g require AO-level knowledge. To propagate conflict information detected by OT to the AO level, corresponding extensions have been made to both OT and TA. With these extensions, OT and TA can collaborate to detect and resolve conflicts among *Group* AO_g. Moreover, desirable combined effects for compatible graphics editing operations can also be achieved with a series of AO-level adjustment strategies.

Chapter 6

Supporting Workspace Awareness in TA-Based Systems

Workspace awareness is particularly important for improving the usefulness of TA-based systems, because it provides the user with the current situation of other collaborators in the unconstrained collaboration environment. This chapter discusses technical issues in supporting workspace awareness in TA-based systems.

6.1. Introduction

Workspace Awareness (WA) is essential for groupware systems. TA-based collaborative systems (e.g. CoWord and CoPowerPoint) have particularly high demands on WA because of the following reasons. First, the workspace of a TA-based system, namely the shared document, may contain numerous data objects in complex structures. Second, TA-based systems allow geographically distributed users to concurrently edit any objects and view any parts of shared documents at any time, which results in constant changes to the workspace. Without effective WA support, it is very difficult for users to perceive others' interactions with this spacious, complex and dynamic workspace.

Collaborative editing activities are centred on the workspace, the shared document, so TA-based collaborative systems have similar WA requirements to other groupware systems. Widely used WA features, including the telepointer

(Crowley et al. 1990), radar view (Gutwin et al. 1996b) and multi-user scroll bar (Roseman and Greenberg 1996), are able to deliver such WA information and thus are suitable for TA-based collaborative systems.

Software reuse has proved to improve software quality and productivity (Basili et al. 1996). It is necessary to design a reusable WA framework for multiple TA-based systems. In addition to reducing the development effort of existing WA features, this WA framework should also facilitate the development of new WA features. This is because TA may be applied to a wide range of single-user applications with drastically different functionalities and interface features, and requires varieties of WA support, some of which may be beyond the capabilities of existing WA techniques and can only be supported by new WA features. To achieve this goal, the WA framework needs to address two technical issues: *object association* and *graphics representation*.

For users to obtain meaningful WA information, WA widgets are usually associated with workspace objects. Existing WA techniques adopt *static* object association schemes in the sense that the object identifier does not change. For example, while the telepointer refers to a window component, it is associated with the target component identifier, which never changes. While referring to a character in a text viewing component, the telepointer is associated with a constant identifier of this component plus a constant index of the character in the text buffer. The invariable object identifiers ensure the correctness of the static reference scheme in a range of groupware systems.

Unfortunately, the static association scheme does not work in TA-based real-time collaborative systems. This is because in such systems, users can edit any objects in the shared document at any time. As a result, positional references of content objects are subject to dynamic changes. These changes may cause problems under two circumstances. First, when a WA widget (e.g. a telepointer) is about to relocate to a new associated object in response to a remote user's action (e.g. mouse cursor movement), the object may have been moved by

concurrent editing operations, which causes the widget to be located at an incorrect position. Second, after a WA widget is relocated, the associated object may be moved by subsequent editing or view changing actions, which may also cause the widget to refer to an incorrect position. The reason of these problems is that existing techniques associate WA widgets with objects' positional references, rather than the objects themselves, so they cannot accommodate the dynamic changes. To solve this problem, a dynamic object association scheme is needed to accommodate the dynamic workspace changes.

Another challenge related to object association is that different object association schemes are adopted by different WA features. For example, a telepointer is used to refer to a specific point in the workspace, so associating it with a single object is sufficient. In a text-based editor, a view port of the radar view should cover the whole view range of a remote user, which is determined by the two objects existing at the view boundaries, so it should be associated with those two objects. Moreover, the telepointer is usually displayed in the main document view, so it should be associated with objects displayed in the main view, while the view port is usually displayed in a miniature document view, so it should be associated with objects displayed in the miniature view. To address this challenge, the object association scheme in the WA framework must be generic enough to accommodate these differences.

Graphics representation is another important technical issue that the WA framework should address. This is because WA features represent WA information by means of graphic widgets. Due to the differences among the WA information types (e.g. presence, location and activity, see Chapter 2), different WA features are represented in different ways. For example, a telepointer is usually represented as an arrow attached with a user name, while a radar view is usually represented as a miniature document view with rectangular view ports. In existing systems, there is no generic graphics representation technique that is able to accommodate these differences.

In this chapter, an innovative technique called *Multi-functional wOrkspace Awareness Framework* (MOAF) is presented. This framework includes an object association technique and a graphics representation technique, which are able to meet the object association and graphics representation requirements of different WA features. Moreover, the MOAF object association technique solves the static object association problem by really associating WA widgets with workspace objects, rather than their positional references. This framework is application-independent, so it can be reused in multiple TA-based collaborative systems. Finally, MOAF not only supports existing WA features, but also can be extended to support new ones.

The rest of this chapter is organized as follows. First, existing object association schemes and graphics representation techniques of WA features are reviewed in Section 6.2. Next, the MOAF object association technique for achieving the object association effects is discussed in Section 6.3. In Section 6.4, the MOAF graphics representation technique is discussed. Afterwards, examples of supporting WA features with MOAF are presented in Section 6.5. Finally, this chapter concludes with a summary of contributions in Section 6.6.

6.2. Related Work

This section reviews existing object association schemes and graphics representation techniques used for WA features.

6.2.1. Existing Object Association Schemes

In existing groupware systems, different object association approaches have been invented to support WA features. For example, multiple object association schemes have been adopted during the evolution process of the telepointer technique. In early generic application-sharing systems, such as CoLab (Stefik et al. 1987) and MMConf (Crowley et al. 1990), the telepointer is displayed at the same position in the shared window. In other words, the telepointer is associated

with the shared window. With the strict WYSIWIS view mode adopted in these systems, all users have the same view of the shared window. This ensures that each object is placed at exactly the same position in the shared window, and the same coordinates point to the same object at all sites, so the window coordinates are sufficient for a telepointer to locate any objects in the shared window.

In a relaxed WYSIWIS view mode, a shared window can have different layouts among participating sites. To accommodate the view difference, techniques associating telepointers with components inside windows have been proposed, including Smart Telepointer (Rodham and Olsen 1994) and GroupKit (Roseman and Greenberg 1996). With these techniques, a telepointer is associated with identifiers of a user interface (UI) component in the shared window, and is provided with the relative position inside the component space. For example, in Smart Telepointer, the telepointer's reference parameters include (1) a path in the component tree from the root to the leaf-level component that contains the telepointer, and (2) the relative position information within the leaf-level component.

Some UI components have internal structures or contents (e.g. a text editor or an HTML viewer). In a relaxed WYSIWIS view mode, the internal content may be formatted and displayed differently due to different view customizations among collaborating sites. For such components, the component-level association is not enough. Smart Telepointer associates the telepointer with the content object position by attaching the index of the associated object (e.g. the character index in a text buffer) in the telepointer reference parameters, so that the telepointer can point to the same content object as the local cursor does. This technique is also adopted in GroupWeb (Greenberg and Roseman 1996).

Similarly, other WA features in existing collaborative systems have their specific object association schemes. For example, the multi-user scrollbar in Groupkit displays multiple scroll boxes to indicate remote users' scroll box locations in a shared scrollbar. The scrolling WA information is collected from

the remote user's scroll shaft and is interpreted based on the current position and size of the local scroll shaft, so the multi-user scroll boxes are associated with the scroll shaft. On the other hand, the Groupkit radar view collects the view awareness information from remote users' scrollbars and interprets this information in the local miniature view. So, the view ports in the radar view are associated with both the (remote) scroll bar and the (local) miniature view window.

Although these object association schemes work well in their own environments and could achieve the effects they were designed for, they are not suitable for TA-based real-time collaborative systems due to dynamic content and view changes. Moreover, these object association schemes are designed for specific WA features. No existing work has been found in the literature that provides generic object association mechanisms for different WA features.

6.2.2. Existing Graphics Representation Techniques

Graphics representation techniques used in existing WA techniques can be generally classified into two categories. The first one is called *direct window-drawing*, which directly draws WA widgets in the underlying workspace window in an XOR mode. When a WA widget (e.g. a telepointer) moves, it is erased from its current position and redrawn to the new position. This approach has been adopted in GroupSketch, GroupDraw (Greenberg et al. 1992) and Dialogo (Lauwers and Lantz 1990). This technique is error-prone because WA widgets have to compete for the drawing area with other functional modules responsible for the graphics representation of the workspace. Since WA widgets are drawn in the same window with the document view, it is difficult to prevent them from interfering with each other and to guarantee the proper display of both sides.

To avoid problems of the direct window-drawing approach, later systems adopt another technique called *glass pane*, which creates a transparent window on top

of the workspace and draws WA widgets on it. This approach has been adopted in GroupKit (Roseman and Greenberg 1992) and MAUI (Hill and Gutwin 2003). Since WA widgets are drawn in a window separated from the workspace, they do not interfere with each other and problems of the direct window-drawing are hence avoided. However, the glass pane window inevitably intercepts all mouse input events to the workspace because it covers the whole workspace area. For the user to manipulate the workspace as usual, mouse input events must be replayed to original target windows in the underlying workspace. In addition to the performance degradation, replaying events properly involves many nontrivial tasks including finding the correct target window and modifying event parameters, which should have been done by the operating system if the glass pane were absent. Therefore, this approach has to end up with a reinvention of the operating system's event dispatching mechanisms. Furthermore, a common problem of the above two approaches is that the movement scope of WA widgets is restricted. The direct window-drawing approach restricts WA widgets within the workspace window; and the glass pane approach restricts WA widgets within the glass pane window (Hill and Gutwin 2003).

6.3. The MOAF Object Association Technique

This section discusses the MOAF object association technique. This technique not only accommodates the dynamic changes in TA-based real-time collaborative systems, but also meets the object association requirements of different WA features.

6.3.1. Object Association Effects

First of all, the desirable object association effects that this technique should achieve are defined. In the following discussion, the telepointer will be used as an example to define the object association effects.

Positional Reference Adjusting (PRA) Effect

In a TA-based system, associated objects are identified with their positional references in the document. A WA widget should be able to adjust the positional references in order to track the associated objects in the face of dynamic content changes caused by editing operations.

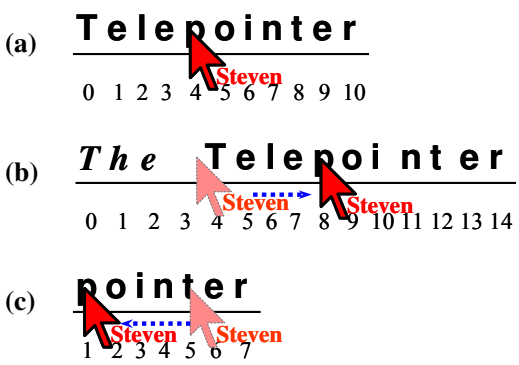


Figure 6.1 The PRA effect (the telepointer tracks the reference character "p"). (a) The initial state; (b) The state after executing an insert; and (c) The state after executing a delete.

Examples of the PRA effect are shown in Figure 6.1. At the initial state (Figure 6.1-(a)), the telepointer is pointing to the character "p" at position 4. After the execution of an insert or delete operation, the positional reference of the character may be changed. To achieve the PRA effect, the telepointer positional reference must be adjusted so that it still points to the character "p", as shown in Figure 6.1-(b) and (c). It should be pointed out that editing operations could be generated concurrently with or sequentially after a telepointer moving operation. The PRA effect must be achieved under both circumstances.

Relative Position-Preserving (RPP) Effect

The WA widget position relative to the associated object should be preserved in the face of dynamic changes to the document. An example of the RPP effect is shown in Figure 6.2. At the initial state (Figure 6.2-(a)), the telepointer is pointing at the centre of the picture. After the execution of a *size-updating* operation, the picture is resized to a quarter of the original size. To achieve the RPP effect, the

telepointer position must be adjusted to accommodate the effect of the updating operation on the object so that it still points to the centre of the picture (Figure 6.2-(b)).

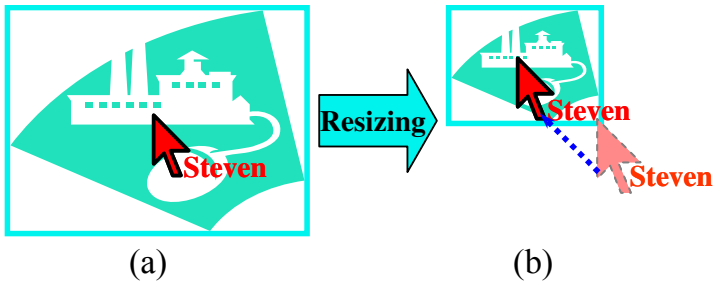


Figure 6.2 The RPP effect. (a) The initial state; (b) The state after executing a resize operation: the telepointer remains inside the picture.

When the user is performing gestures with the mouse cursor, the cursor is more often outside rather than inside the associated object. The RPP effect should also be achieved when the telepointer is outside the associated object or in a blank area. In this case, the telepointer is associated with the nearest object.

An example is shown in Figure 6.3. In the initial state (Figure 6.3-(a)), the telepointer is in the blank area near the picture. After the picture is resized, the telepointer is relocated accordingly so that it still points at the same position relative to the associated object (Figure 6.3-(b)).

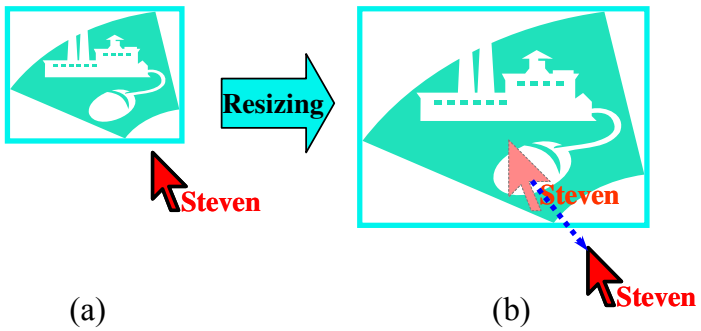


Figure 6.3 The RPP effect when the telepointer is in a blank area. (a) The initial state; (b) the state after executing a resize operation on the picture: the telepointer remains outside the picture.

The Local WA Widget

Telepointers are used to represent the positions of their corresponding local cursors, and therefore they should be kept consistent with the local cursors. To achieve the PRA and RPP effects, telepointers may be relocated dynamically to track the associated objects. After the relocation of the telepointers, the positions of these telepointers at remote sites may no longer be consistent with their corresponding local cursor.

One way to keep them consistent is to relocate the local cursor as well, but this can be disruptive to the user. To solve this problem, the notion of a local WA widget, which is the same as a WA widget but displayed at the local site, is introduced. For the telepointer, the local WA widget is a virtual local cursor. When relocation of the telepointer occurs at a remote site, the virtual local cursor will be relocated to track the associated object, but the local mouse cursor is not moved. This virtual local cursor provides feedback to the local user about the locations of his/her telepointers at remote sites.

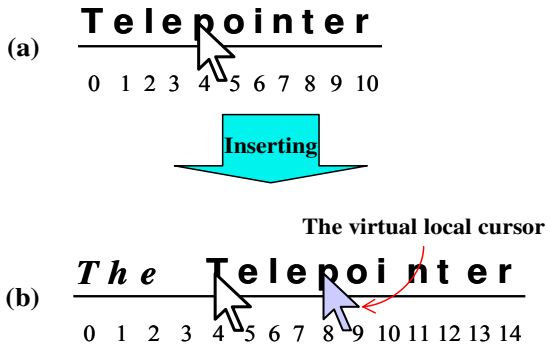


Figure 6.4 The virtual local cursor for tracking the associated object. (a) The initial state; (b) the state after the associated object is pushed to the right. The virtual local cursor follows the associated object, but the real local cursor is not affected.

Consider the example shown in Figure 6.1. When the string "The " is inserted, the telepointer is relocated to track the character "p" (Figure 6.1-(b)). What happens at user Steven's local site at the same time is shown in Figure 6.4. In the initial state (Figure 6.4-(a)), the local cursor is pointing at the character "p" at

position 4. After the string is inserted, the virtual local cursor appears and tracks the associated object (Figure 6.4-(b)), but the real local cursor is not affected.

When the user moves the local cursor, the virtual cursor should disappear. While moving the local cursor, the user may intend to point to another object. In this case the new associated object should be identified and associated with the telepointers, and remote telepointers should be relocated accordingly. The user may also want to move the local cursor to point to the original associated object. With the virtual local cursor pointing to the original associated object, it is much easier for the user to find the moved object from the documents.

Discussion

There are some issues related to the above object association effects that are worth discussing.

To guarantee the correctness of gesturing WA features (e.g. the telepointer), one alternative to the dynamic object association is to prohibit all users from editing during the (non-deterministic) gesturing period. It may seem natural that the gesture-generating and accepting users would not edit during the gesturing period. However, in the same collaborative editing session, not all users are interested in the gestures. Prohibiting everyone from editing while someone is gesturing is undesirable. Another alternative is to require the gesture-generating user to wait until others have stopped editing. This solution is also undesirable because the occurrence of such a quiescent moment is unpredictable. Even if there are only two users (i.e. the gesture generator and accepter), it will be beneficial if they have the convenience to gesture and edit at the same time without any extra effort.

The degrees to which different WA features achieve the object association effects are also different, because different WA features have different behavior and characteristics in response to these changes. For example, the telepointer is often used as a gesturing tool. It requires high precision and smooth movement,

so it needs to achieve the RPP effect. On the other hand, the radar view is used to indicate which objects a remote user can see. One seldom needs to know precisely which part of a character at the view boundary a remote user can see. In this sense, achieving the RPP effect is not necessary for the radar view.

The local widget of different WA features also behaves differently. The virtual local cursor for the telepointer appears only when the local mouse cursor is inconsistent with its telepointers. However, the local view port of the radar view is always displayed in the miniature view, because it provides the local user with location information about where his/her view range is in the global workspace.

6.3.2. Adapting Workspace Awareness AO

Like other TA-based collaboration techniques (e.g. CoTable and CoGroup in Chapter 5), the MOAF object association technique is also supported by a set of AO and corresponding adaptation techniques.

The Workspace Awareness AO Definition

To carry object-associated WA information among distributed collaborating sites running the TA-based collaborating editing system, a set of *Workspace Awareness AO*, called AO_w are defined. When any event that changes the workspace state occurs at the local site, an AO_w is generated and propagated to all remote sites. When a collaborating site receives an AO_w, it interprets the WA information contained in this AO_w and updates the position or shape of the corresponding WA widget to reflect the WA information encapsulated in the AO_w. The MOAF object association technique is applied to guide the AO_w processing.

As a polygon, the position and shape of a WA widget are determined by its vertices, so AO_w need to carry information about positions of all vertices of the WA widget. The following two AO_w are defined to carry such WA information.

(1) *MoveAO_w(wa_type, vertex$_1$)*

(2) *ReshapeAO$_w$(wa_type, vertex$_1$, vertex$_2$ ... vertex$_n$)*

The *wa_type* parameter is the identifier of the WA feature type. A *vertex* parameter contains information for calculating the X and Y coordinates of a vertex, so it is defined as follows.

vertex ((obj_id_x, offset_x), (obj_id_y, offset_y))

The parameter *obj_id_x* is the identifier of the object associated with the X coordinate of the vertex. When a vertex refers to a data object in the document, this parameter is the object's positional reference in the document data model (i.e. a vector of *(n, p)* pairs, see Chapter 4). When the vertex refers to a UI object (e.g. a button), this parameter is the UI object's globally unique identifier. The parameter *offset_x* is the horizontal distance between the vertex and the associated object's left edge. This distance is measured as a relative ratio to the width of the associated object, rather than an absolute pixel number. For example, an *offset_x* value 0 indicates the X coordinate value of the associated object's left edge; and an *offset_x* value 1 indicates the X coordinate value of the associated object's right edge.

Parameters *obj_id_y* and *offset_y* are defined similarly but are used to calculate the vertex's Y coordinate. With these parameters and the current status of the associated object, coordinates of a vertex can be calculated with the following formulae:

$$vertex.x = obj\_x.left + obj\_x.width \times offset\_x$$
$$vertex.y = obj\_y.top + obj\_y.height \times offset\_y$$

MoveAO$_w$ is used to change the position of a WA widget without affecting its shape, so it contains information for calculating the position of the first vertex only. *MoveAO$_w$* is suitable for WA features whose shapes never change, such as the telepointer. *ReshapeAO$_w$* is used to change both the position and shape of a WA widget, so it contains information for calculating positions of every vertex.

Moreover, the vertex number encapsulated in the AO_w may be different from the current vertex number of the WA widget. That means the vertex number of a WA widget may also be changed upon receiving a *ReshapeAO*$_w$. *ReshapeAO*$_w$ is suitable for WA features whose shapes may change as well as positions, such as the view port.

To generate an AO_w when the workspace state is changed, WA information needs to be represented in the context of workspace objects (may be data objects in the document or UI objects). For WA information originally represented in absolute screen positions (e.g. the mouse cursor position), a translation process is required. Typically, the translation process involves the following two steps.

(1) First, workspace objects that are suitable for the association are identified. For every coordinate value contained in the WA information, one object is identified.
(2) Next, the relative horizontal and vertical positions are calculated according to the current status (e.g. positions and sizes) of associated objects.

In the TA framework, techniques to process AO_w can be designed in the same way as processing techniques for other AOs. The desirable object association effects can be achieved by processing AO_w, as discussed in the following.

Adapting Data Object-Referring AO_w in the TA Framework

Objects referred by AO_w could be UI objects or data objects. In the former case, static global identifiers of target UI objects can satisfy the identification needs because such identifiers are never affected by content and view changes. In the latter case, however, AO_w need to refer to target data objects with their XOTDM addresses like editing AOs. Therefore, concurrency-related inconsistency problems that may happen to editing operations may also happen to these data object-referring AO_w. To handle these problems, data object-referring AO_w should also be adapted in the TA framework so that they can be transformed by OT. On the other hand, unlike editing AOs, AO_w never change the state of target

objects. Instead, they only refer to objects. This characteristic cannot be captured by the three POs supported by OT, so it is not suitable to translate data object-referring AO_w into any existing PO types.

To address this issue, a new PO type, called *Refer* is defined as follows:

Refer(vp) denotes referring to the object at the position *vp*.

The *vp* parameter is the same one as in the *Insert*, *Delete* or *Update* POs, which is a vector of (*n, p*) pairs to indicate the address of the associated object in the document data model.

Data object-referring AO_w are compound AOs because one AO_w should be translated into multiple *Refer* POs. When translating an AO_w into *Refer* POs, the *obj_id_x* and *obj_id_y* of every *vertex* parameter are individually passed to a *Refer* PO as the *vp* parameter. In this way, every *vertex* parameter is translated into two *Refer* POs.

Extending OT to Transform Refer

OT is able to transform three PO types, which do not include *Refer*. To support transforming *Refer*, OT needs to be extended correspondingly.

As discussed in Chapter 4, for supporting a new PO type, a set of transformation functions for the new PO type should be designed, and the high-level transformation control layer should be kept unchanged. Since *Refer* does not have effects on other operations, ET functions for *Refer* are not needed. For the same reason, IT functions that transform other POs against *Refer* are not needed either. Therefore, only IT functions for transforming *Refer* against other POs need to be designed. Finally, mechanisms for processing operations targeting different linear addressing domains are encapsulated in the VOT function (see Chapter 4), so only IT functions for transforming operations targeting the same linear addressing domain are needed.

The OT technique supports three primitive operations, which are *Insert*, *Delete* and *Update*. IT functions transforming *Refer* against these operations are shown in Figure 6.5.

```
IT_RI(Or, Oi)
{
   if (Oi.vp[last] <= Or.vp[last])
      Or. vp[last] = Or. vp[last] + Oi.len;
   return Or;
}

IT_RD(Or, Od)
{
   if (Od. vp[last] + Od.len < Or. vp[last])
      Or. vp[last] = Or. vp[last] - Od.len;
   else if ((Od. vp[last] < Or. vp[last]) && (Od.pos + Od.len >= Or. vp[last]))
      Or. vp[last] = Od. vp[last];
   return Or;
}

IT_RU(Or, Ou)
{
   return Or;
}
```

Figure 6.5 IT functions for the *Refer* operation.

When a *Refer* is transformed against an *Insert* (*IT_RI*), the *Refer*'s position is shifted to the right by the *Insert*'s length if the *Insert*'s position is to the left of the *Refer*'s position, because the associated object is pushed to the right. If the *Insert*'s position is to the right of the *Refer*'s position, then the *Refer*'s position parameter is not changed.

Transforming a *Refer* against a *Delete* (*IT_RD*) is more complex. If the range of the *Delete* is completely to the left of the *Refer*'s position, then the *Refer*'s position is shifted to the left by the *Delete*'s length, because the associated object is pulled to the left. If the *Delete*'s range covers the *Refer*'s position, then the position of the *Refer* is set to the position of the *Delete*, because the original associated object is deleted by the *Delete* and the object at the *Delete*'s position (*Od.vp*) becomes the new associated object. Finally, if the *Delete*'s position is to

the right of the *Refer*'s position, then the *Refer*'s position parameter is not changed.

When a *Refer* is transformed against an *Update* (*IT_RU*), the *Refer*'s position parameter is not changed, because an *Update* does not affect the position of the associated object in the data model.

6.3.3. Achieving Object Association Effects

With the AO_w definition and the adaptation technique in TA and OT, the desirable object association effects can be achieved, as discussed in the following.

Handling Concurrent Editing Operations

The major technical challenge of achieving the PRA effect is to locate the associated object whose positional reference has been changed due to concurrent editing operations. With the AO_w processing technique discussed above, the object identifier parameters in an AO_w can be adjusted with OT and TA, so that an AO_w can always locate the associated objects correctly in the face of concurrent editing operations. Therefore, the PRA effect is achieved with the support of OT and TA.

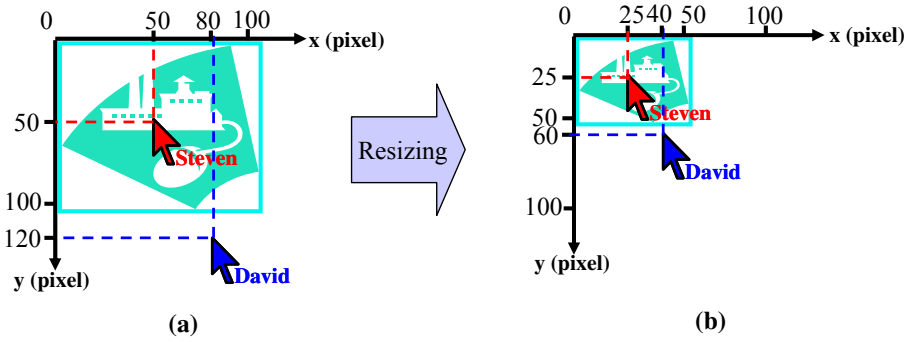


Figure 6.6 A scenario of achieving the RPP effect with the relative ratio position parameters. (a) The initial state; (b) The state after resizing.

The RPP effect can be achieved by making use of the relative ratio position parameters of the AO_w. When the associated object has been found from the

document, coordinates of the new vertex position can be calculated with its current status and the relative ratio position parameters. In this way concurrent attribute changes (e.g. resizing) to the associated object can be accommodated. An example of achieving the RPP effect for the telepointer is shown in Figure 6.6.

In the initial state (Figure 6.6-(a)), the associated object, whose positional reference is *obj_pos*, occupies an area of 100 * 100 pixels. The first vertex of user Steven's telepointer is at the centre of the associated object, so this vertex is specified as ((*obj_pos,* 0.5), (*obj_pos,* 0.5)), corresponding to the relative pixel position <50, 50>. User David's telepointer is outside the associated object and the relative ratio position is ((*obj_pos,* 0.8), (*obj_pos,* 1.2)), corresponding to the relative pixel position <80, 120>. After the associated object is resized to 50 * 50 pixels (Figure 6.6-(b)), positions of the two telepointers are recalculated. Based on the new size of the associated object and the relative ratio positions, the relative pixel position of user Steven's telepointer's first vertex is changed to <25, 25>, so that it is still at the centre of the associated object; the relative pixel position of user David's telepointer's first is changed to <40, 60>, so that it is still at the same position relative to the associated object.

To keep the local WA widget consistent with remote widgets, the PRA and RPP effects should also be achieved while relocating the virtual local cursor. Since the local WA widgets act as mirrors of the remote counterparts, they can be controlled by the same technique for handling remotes widgets.

Handling Subsequent Editing Operations

Apart from concurrent editing operations, subsequent editing operations executed after an AO_w may also change the on-screen position or size of the associated object and hence invalidate the association between the WA widget and its associated objects.

Editing operations executed at any address could affect the position or size of the associated object. First, operations targeting the associated object could

directly change its position or size. Second, operations executed before an associated object (in the data model) could change its address in the data model and thus affect its on-screen position (see Figure 6.1-(b) and (c)). Finally, operations executed after the associated object could change the layout of the document view, and hence indirectly affect the associated object's on-screen position.

To solve these problems, the following *WA widget relocation scheme* for accommodating changes caused by subsequent editing operations is devised.

(1) Addresses of all associated objects are adjusted to accommodate the effect of the subsequent editing operation. This adjustment can be done by transforming the latest AO_w of each WA widget against the editing operation (Xia et al. 2005c).
(2) New positions of all WA widgets vertices are recalculated based on the current status of associated objects.
(3) WA widgets are moved or reshaped to new positions if necessary.

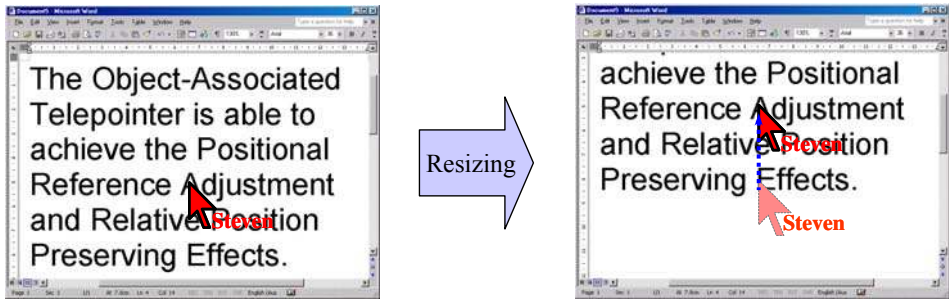
It should be pointed out that this relocation scheme is also applied to the local WA widgets so that it can also achieve the object association effects in the face of dynamic changes caused by subsequent editing operations.

Handling View Changes

WA widgets are displayed in a layer different from the document view windows (to be discussed in the graphics representation technique in the next section). Therefore, view changes (e.g. scrolling up and down, zooming in and out) could also affect the association between WA widgets and associated objects. An example is shown in Figure 6.7.

In the initial state (Figure 6.7-(a)), the telepointer is pointing at the centre of the character "A". After the document view is scrolled up, the position of the associated object, character "A" is moved. To preserve the PRA and RPP effects,

the telepointer needs to be moved up as well so that it still points at the centre of the character "A" (Figure 6.7-(b)).



(a) **(b)**

Figure 6.7 A scenario for preserving the object-associated effects in the face of view change. (a) The initial state; (b) The state after scrolling.

Like editing operations, view changes could also be concurrent with or sequential to the AO_w. The effect of the concurrent view change is accommodated in the vertex position calculation process. When an AO_w operation is executed at a remote site, the current status (after the view change) of the associated object is used to calculate the vertex positions. In this way, the WA widgets position has taken the effect of view changes into account.

View changes affect the position and size of the associated objects without changing their internal state in the document. Therefore, when a subsequent view change occurs, addresses of associated objects in the data model do not need adjustment. Only the vertex position recalculation and WA widgets relocation are needed. For the example shown in Figure 6.7, after the view has been scrolled up, the new status of the associated object the character "A" is obtained first. Then the new on-screen position of the telepointer is calculated based on the current status of the associated object and the relative position parameters of the telepointer. Finally, the telepointer is moved to the new position.

6.4. The MOAF Graphics Representation Technique

This section discusses the MOAF graphics representation technique. To accommodate varieties of graphics demands of different WA features, the MOAF graphics representation technique should meet the following requirements.

(1) It should support creating and maintaining graphic objects with all attributes needed by different WA features, including shape, position, filling color and semi-transparency.
(2) It should support easy manipulation of graphic objects. WA widgets are frequently moved or reshaped. It is important that the mechanism to manipulate related attributes is simple and efficient.
(3) Graphics representation of WA widgets should be independent of the shared workspace. Attribute changes of WA widgets should not interfere with the workspace display.

In the MOAF graphics representation technique, windows, the basic *Graphic User Interface* (GUI) element in windowing platforms, are used as the graphics representation means of WA widgets. This approach takes advantage of the GUI functionalities of windowing platforms (e.g. Microsoft Window, X Windows, and Mac OS), which support windows in any non-rectangular shapes. With the support of the graphics functionalities of the windowing platform, window-based WA widgets can be created with customized shapes and other graphic attributes to meet the graphic demands of different WA features. Moreover, graphs can be drawn in these windows to provide more detailed WA information. For example, a telepointer can be represented as an arrow-shaped window attached with a text string; and a view port can be represented as a semitransparent rectangular window with a text string in it.

With this approach, control of WA widget windows is simple. Graphic attributes of WA widget windows can be easily manipulated with the windowing

platform APIs. For example, moving a WA widget requires only one API call in MS Windows. On the contrary, with the direct window drawing or the glass pane approach (see Chapter 6.2), moving a WA widget involves complex tasks including erasing the widget from its current position and redrawing it at the new position. Moreover, window-based WA widgets are able to move around the whole screen without any limitation.

6.5. Supporting WA Features with MOAF

With the MOAF technique, the main tasks for supporting a WA feature include analysing the relationship between WA widget vertices and workspace objects, and defining the corresponding AO_w to express this relationship. In this section, how these tasks are performed will be illustrated with examples.

6.5.1. Radar View

As shown in Figure 6.8, a view port of the radar view is a semitransparent rectangle covering all objects a remote user can see, so the AO_w for the radar view should contain position information of the four vertices of the view port rectangle, including their associated object identifiers and offset values, as shown in the AO_w definition below:

ReshapeAO_w (RADAR_VIEW, V1, V2, V3, V4), in which *RADAR_VIEW* is the WA type identifier of the radar view, and the remaining parameters encapsulate object-associated vertex positions.

Two different object association schemes can be adopted for the radar view. In the user interface of a typical graphics-based editor (see Figure 6.8-(a)), graphic objects are placed in a drawing canvas, which is the global coordinates space for all distributed collaborating sites. A user's view of the workspace is a segment of the global canvas. Positions of graphic objects are defined in this canvas, so the

canvas is an ideal associated object for the view ports. Therefore, coordinates of all vertices are associated with the canvas object (in the miniature document view). The vertex parameters of the graphics-based radar view AO_w are represented as:

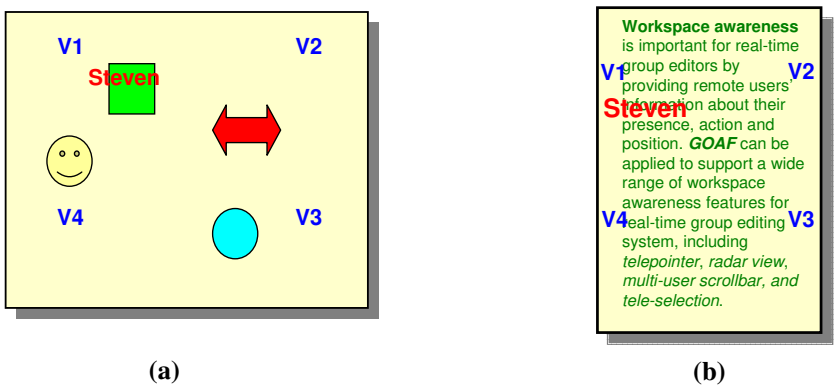


Figure 6.8 The radar views. (a) The radar view of a graphics-based editor; (b) the radar view of a text-based editor.

V1 = ((MINIATURE_CANVAS_ID, offset_1_X), (MINIATURE_CANVAS_ID, offset_1_Y)),

V2 = ((MINIATURE_CANVAS_ID, offset_2_X), (MINIATURE_CANVAS_ID, offset_2_Y)),

V3 = ((MINIATURE_CANVAS_ID, offset_3_X), (MINIATURE_CANVAS_ID, offset_3_Y)),

V4 = ((MINIATURE_CANVAS_ID, offset_4_X), (MINIATURE_CANVAS_ID, offset_4_Y)),

in which *MINIATURE_CANVAS_ID* is the identifier of the canvas object in the miniature document view, and *offset_i_x* and *offset_i_y* indicate the horizontal and vertical position of the *i*th vertex relative to the left top corner of the canvas object.

In the user interface of a typical text-based editor, such a global canvas object does not usually exist. However, user interfaces of these applications have another characteristic – data objects (namely characters) are represented to users in a linear sequence. So, a user's view can be specified by the two objects existing on the upper and lower view boundaries.

Figure 6.8-(b) shows the object association scheme for the view port in a text-based editor. The left and right edges of the view port overlap with the left and right edges of the miniature document view window. The top and bottom edges of the view port overlap with the top and bottom edges of the data objects existing at the view boundaries. Therefore, vertex parameters of the text-based radar view AO_w are represented as:

V1 = ((MINIATURE_VIEW_ID, 0), (obj_1, 0)),

V2 = ((MINIATURE_VIEW_ID, 1), (obj_1, 0)),

V3 = ((MINIATURE_VIEW_ID, 1), (obj_2, 1)),

V4 = ((MINIATURE_VIEW_ID, 0), (obj_2, 1)),

in which *MINIATURE_VIEW_ID* is the identifier of the miniature document view window; *obj_1* and *obj_2* are the positional references (in the data model) of data objects existing at the upper and lower boundaries of the main document view.

The data object-based association scheme for text-based editors can also be applied in graphics-based editors, because graphic objects can also be accessed with their positional references in the data model (see Chapter 3 and Chapter 5). This example illustrates that multiple object association schemes are applicable while supporting WA with MOAF. Developers of TA-based systems may make their decisions based on the characteristics of concrete applications. For example, the data object-based association scheme is adopted in CoWord due to the absence of the global canvas, although Word documents also contain graphic objects.

6.5.2. Telepointer

The shape of a telepointer never changes, so $MoveAO_w$ is chosen as the AO_w for the telepointer, which is defined as follows:

$MoveAO_w$ *(TELEPOINTER, V1)*, in which *TELEPOINTER* is the WA type identifier of the telepointer, and *V1* encapsulates the object-associated position of the first vertex. *V1* is defined as:

V1 = ((obj, offset_x), (obj, offset_y)),

in which *obj* is the identifier of the workspace object nearest to the remote user's mouse cursor. This object can be either a data object in the document or a UI object because the mouse cursor is free to move around the workspace. In the former case, the *obj* parameter is the positional reference (in the data model) of the data object. In the latter case, the *obj* parameter is a global identifier of the UI object. The *offset_x* and *offset_y* parameters are horizontal and vertical positions of the mouse cursor relative to the left top corner of the associated object (as shown in Figure 6.9).

Figure 6.9 The telepointer.

6.5.3. Multi-User Scrollbar

As shown in Figure 6.10, a scroll box of the multi-user scrollbar is a rectangle whose position and size are determined by its four vertices, so the multi-user scrollbar AO_w should carry information to calculate positions of the four vertices, as defined in the following.

ReshapeAO$_w$ *(MULTIUSER_SCROLLBAR, V1, V2, V3, V4),* in which *MULTIUSER_SCROLLBAR* is the WA type identifier of the multi-user scrollbar, and the remaining parameter encapsulate object-associated vertex positions.

The multi-user scrollbar collects the scroll box position information from the remote scroll shaft and interprets this information in the local scroll shaft, so positions of its vertices should be associated with the scroll shaft object. To realize this association, the vertex parameter of the multi-user scrollbar AO_w should be defined as the following:

V1 = ((SCROLL_SHAFT_ID, 0), (SCROLL_SHAFT_ID, offset_top)),

V2 = ((SCROLL_SHAFT_ID, 1), (SCROLL_ SHAFT _ID, offset_top)),

V1 = ((SCROLL_SHAFT_ID, 1), (SCROLL_ SHAFT _ID, offset_bottom)),

V1 = ((SCROLL_SHAFT_ID, 0), (SCROLL_ SHAFT _ID, offset_bottom)),

in which *SCROLL_SHAFT_ID* is the identifier of the scrollbar shaft; the *offset_top* is the top position of the scroll box relative to the shaft; and the *offset_bottom* is the bottom position of the scroll box relative to the shaft.

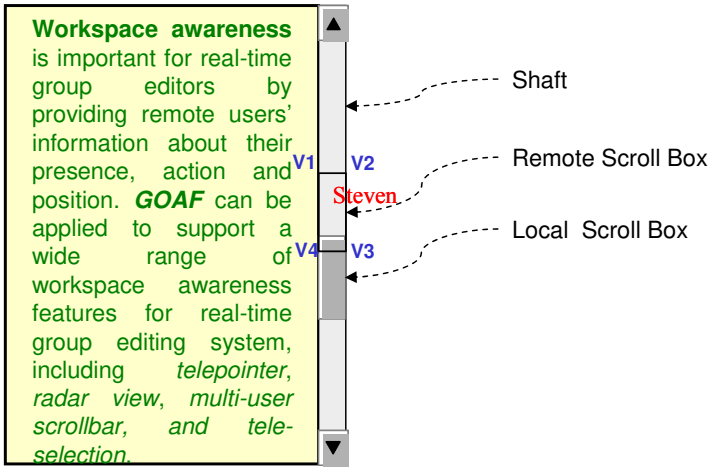


Figure 6.10 The multi-user scrollbar.

6.5.4. Teleselection

The last example is a new WA feature called *teleselection*, which indicates a remote user's selection range in the shared document. When the remote user does

not select anything, it is hidden in graphics-based editors and degrades to a tele-caret in text-based editors. This WA feature is able to deliver rich WA information including: (1) location, since it indicates where the remote user is working; (2) activity, since the user's actions can only happen in his/her selected range; and (3) intention, since the user has to select the target objects before manipulating them. Moreover, it can also be used as a collaborative highlighting tool (Shen and Sun 2004) to facilitate gesturing and communication. The teleselection feature has not been seen in existing groupware systems. This example demonstrates the flexibility and extensibility of MOAF.

The teleselection widget is designed as semitransparent polygons covering all data objects selected by the user. As shown in Figure 6.11, teleselection widgets may have different shapes in different applications. In a graphics-based editor (shown in Figure 6.11-(a)), the user can select multiple discrete objects, so the teleselection widget for a remote user should be represented as multiple rectangles, each covering one selected object.

To support the multiple rectangular teleselection widgets in graphics editors, the AO_w is defined as follows:

ReshapeAO$_w$ (TELESELECTION, V1_1, V1_2, V1_3, V1_4, V1_5, V2_1 ...), in which *TELESELECTION* is the WA type identifier of the teleselection, and *Vi_j* contains the object-associated position information of the *j*th vertex of the *i*th rectangle.

Vertex parameters of the *i*th rectangle are defined as follows:

Vi_1 = ((obj_i, 0), (obj_i, 0)),

Vi_2 = ((obj_i, 1), (obj_i, 0)),

Vi_3 = ((obj_i, 1), (obj_i, 1)),

Vi_4 = ((obj_i, 0), (obj_i, 1)),

Vi_5 = ((obj_i, 0), (obj_i, 0)),

in which *obj_i* is the identifier of the *i*th selected object.

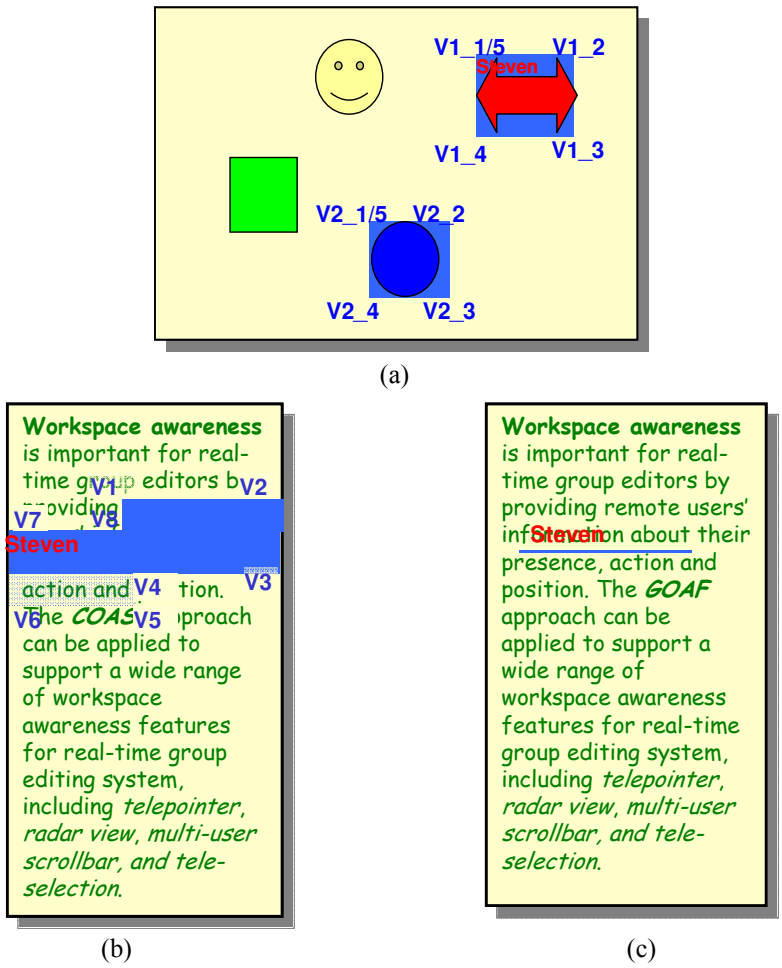


Figure 6.11 The teleselection. (a) The teleselection of a graphics-based editor; (b) the teleselection of a text-based editor (multiple lines); (c) the teleselection of a text-based editor (single line).

It should be noted that 5 vertices are defined to specify the region of a rectangle. The fifth vertex overlaps with the first one so that these vertices define a close region of the rectangle. With this special treatment, rectangles are separated from each other.

Unlike the radar view for graphics-based editors, the teleselection is associated with selected graphic objects, rather than the canvas object. This is because editing operations concurrent with the user's selecting actions may change the size or position of the selected object. When a teleselection AO_w is executed, the

selected objects may have already been moved or resized by concurrent operations, and the teleselection widget may fail to cover them if it is associated with a static UI component (i.e. the canvas). By associating the teleselection with data objects with positional references in the data model, the teleselection widget can always be placed at the right position, thanks to the underlying TA and OT techniques.

In a text-based editor (shown in Figure 6.11-(b) and (c)), the user can only select continuous objects, so the teleselection widget is an octagon (shown in Figure 6.11-(b)). Therefore, the AO_w is defined as follows:

ReshapeAO_w (TELESELECTION, V1, V2, V3, V4, V5, V6, V7, V8). Vertex parameters of this AO_w are defined as follows:

V1 = ((obj_1, 0), (obj_1, 0))

V2 = ((MAIN_VIEW_ID, 1), (obj_1, 0)),

V3 = ((MAIN _VIEW_ID, 1), (obj2, 0)),

V4 = ((obj_2, 1), (obj_2, 0)),

V5 = ((obj_2, 1), (obj_2, 1)),

V6 = ((MAIN _VIEW_ID, 0), (obj_2, 1)),

V7 = ((MAIN _VIEW_ID, 0), (obj1, 1)),

V8 = ((obj_1, 0), (obj_1, 1)),

in which *obj_1* and *obj_2* are the identifiers of the first and last selected objects and the *MAIN _VIEW_ID* is the identifier of the main document view window.

A special case is that when the selection is within one line (shown in Figure 6.11-(c)), the teleselection widget is a rectangle, rather than an octagon. However, this case is also covered by the above AO_w definition. When *obj_1* and *obj_2* are in the same line, their top and bottom boundaries are equal. So, vertices *V2* and *V3* merge to one vertex and so do vertices *V6* and *V7*. In this way, the octagon degrades to a rectangle. An alternative solution is to define the AO_w in this case

as a rectangle with four vertices. The disadvantage of this solution is that it complicates the AO_w generation process since it has to distinguish two cases. More importantly, even if the boundary objects are in the same line at the local site, they may be in different lines at remote sites because of the view differences. This solution cannot accommodate this case. Therefore, the unifying solution was chosen to accommodate both cases.

6.5.5. Discussion

Similar to the *glass pane* technique, our WA widgets are placed on top of the workspace. The problem of intercepting the user's input event could also occur in MOAF. In this subsection, we discuss issues and methods to alleviate or avoid this problem without reinventing the windowing platform's event dispatching mechanisms.

WA widgets displayed in the miniature view do not interfere with the user's interaction, because the miniature view is not supposed to accept editing operations. In the main document view, small WA widgets are unlikely to interfere with the user's interaction. For example, users seldom mistakenly click the mouse bottom on a telepointer. Therefore, this problem mainly occurs on WA widgets that are displayed in the main view and whose areas are considerably large, such as teleselection widgets. One solution to this problem is to implement them as frames which consist of only a few lines. In this way, the chance of intercepting the user's input events is significantly reduced to a negligible degree. With the graphics capability of windowing platforms and the flexibility of MOAF, supporting frame-shaped widgets is not difficult.

One possible problem from this solution is that the user may have difficulties in seeing the frame-shaped widgets or differentiating them from workspace objects in a crowded workspace. This problem can be solved by adding some attractive features on the widget frames, such as flashing or animation.

6.6. Summary

In this chapter, an innovative MOAF technique to support WA features in TA-based collaborative systems has been discussed.

MOAF contains an object association technique and a graphics representation technique. The MOAF object association technique is able to preserve a series of object association effects in the face of dynamic content and view changes. To achieve this goal, a set of AO_w is defined to carry WA information. Each AO_w contains information about object association parameters of all vertices of the polygonal WA widget. To adjust object reference parameters of AO_w in the face of concurrent and consequent editing operations and view changes, AO_w adaptation techniques are designed so that AO_w can be processed with the underlying OT technique. Meanwhile, the OT technique is extended to support a new PO type *Refer*, which denotes referring to an object in the data model without modifying it. With these techniques, the desirable object association effects can be preserved.

The MOAF graphics representation approach utilizes the GUI functionalities of windowing platforms to represent WA information. The MOAF graphics representation technique is able to meet the graphics representation requirements of different WA features. With this technique, WA widgets do not interfere with the workspace display. Furthermore, creation and manipulation of WA widgets are easier and less error-prone than existing techniques.

Finally, examples of supporting existing and new WA features, including the radar view, telepointer, multi-user scrollbar and teleselection, were presented to demonstrate the feasibility and flexibility of MOAF.

The MOAF technique is able to reduce the effort for developing existing WA features and can be easily extended to support new WA features in TA-based real-time collaborative editors. It has been applied in the CoWord and

CoPowerPoint systems to support multiple WA features, thereby showing its applicability in different applications.

Chapter 7

The CoWord and CoPowerPoint Prototypes

CoWord and CoPowerPoint are two experimental prototype systems produced from this research. These two systems verified the feasibility, effectiveness and generality of approaches and techniques generated from this research. Moreover, they are also useful collaborative editing systems on their own. This chapter discusses the design and implementation issues and initial usage experiences of these two systems.

The rest of this chapter is organized as follows. First, the system architecture of CoWord and CoPowerPoint is described in Section 7.1. Then details of components and modules of CoWord and CoPowerPoint are discussed in Section 7.2. Next, functionalities and user interface features of these two systems are presented in Section 7.3. In Section 7.4, experiences accumulated from the implementation of CoWord and CoPowerPoint are described. Afterwards, initial usage experiences and feedback are discussed in Section 7.5. Finally, this chapter concludes with a summary in Section 7.6.

7.1. A TA-Based Collaborative System Architecture

Based on the TA approach, a system architecture is proposed, as shown in Figure 7.1-(a). This architecture consists of three components:

(1) *Single-user Application* (SA), which provides conventional single-user interface features and functionalities. This component is unaware of multi-user collaboration.

(2) *Collaboration Adaptor* (CA), which provides application-specific collaboration capabilities and plays a central role in adapting SA to the underlying GCE (see below). This component is aware of both single-user application and multi-user collaboration.

(3) *Generic Collaboration Engine* (GCE), which provides application-independent collaboration capabilities. This component encapsulates a package of collaboration-supporting techniques, with OT at the core for supporting consistency maintenance and group undo. This component is aware of multi-user collaboration, but unaware of the single-user application.

The use of CA between SA and GCE hides application-specific issues from GCE, facilitates independent debugging and testing of GCE, and promotes the reusability of GCE. The ability to reuse GCE is important and valuable because the design and implementation of a correct and efficient GCE is challenging due to the complexity involved. To apply GCE to a new SA, one only needs to design and implement a new CA for the target SA.

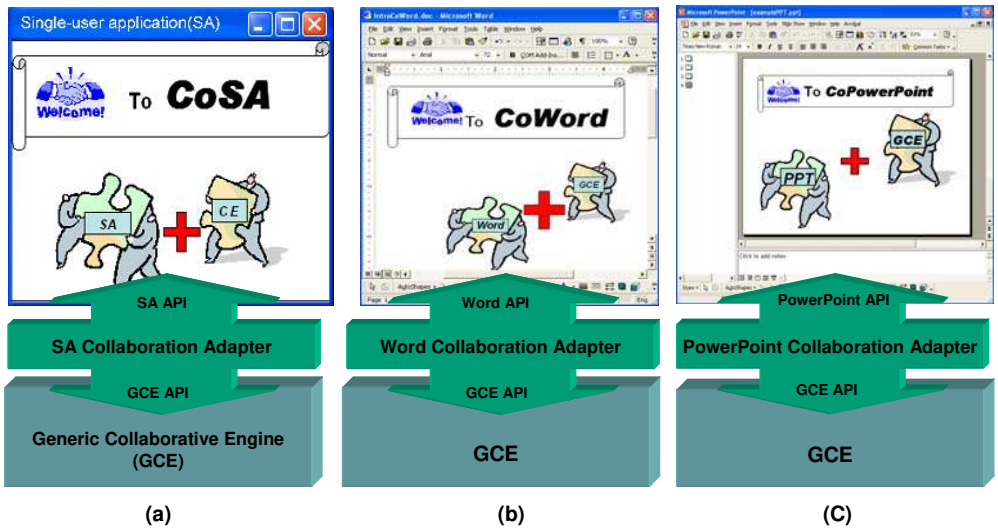


Figure 7.1 (a) A generic collaborative system architecture. (b) CoWord system architecture. (c) CoPowerPoint system architecture.

Based on the generic collaborative system architecture, two working prototype systems, CoWord and CoPowerPoint, have been built to demonstrate the feasibility of the TA approach, the system architecture, and supporting techniques. The architectures of CoWord and CoPowerPoint are shown in Figure 7.1-(b) and (c).

7.2. Components and Modules

Figure 7.2 shows the architecture of CoWord in more details. The architecture of CoPowerPoint is similar, except that the SA component is PowerPoint, rather than Word.

7.2.1. The Collaboration Adaptor

Major modules in the Collaboration Adaptor (CA) component (Figure 7.2) are described as follows.

The *API-AO Adaptation* module is responsible for the interpretation of AOs by means of the application's API, as discussed in Chapter 3. In addition, this module also provides an adapted API for other CA modules to access the application's API, thus hiding the application-specific details from the rest of the system.

The *AO-PO Adaptation* module is responsible for translation between AOs and POs, as discussed in Chapter 3. It also provides a common interface between other CA modules and GCE.

The *Local Operation Handler* (LOH) module is responsible for intercepting the local user interactions and generating corresponding AOs, as discussed in Chapter 3. LOH also controls the granularity of AOs. For example, a sequence of character insertions may be packed into a single string-wise insertion. LOH makes use of the AO-PO Adaptation module to translate the generated AO into suitable POs, which are timestamped and saved in the local history buffer of OT

(inside GCE) for consistency maintenance. Apart from data-manipulation operations, LOH also intercepts non-data-manipulation events generated by the user from the single-user application interface (e.g. moving the scroll bar, resize the window, move the cursor), or from the CoWord/CoPowerPoint Control Panel for interaction control and workspace awareness support.

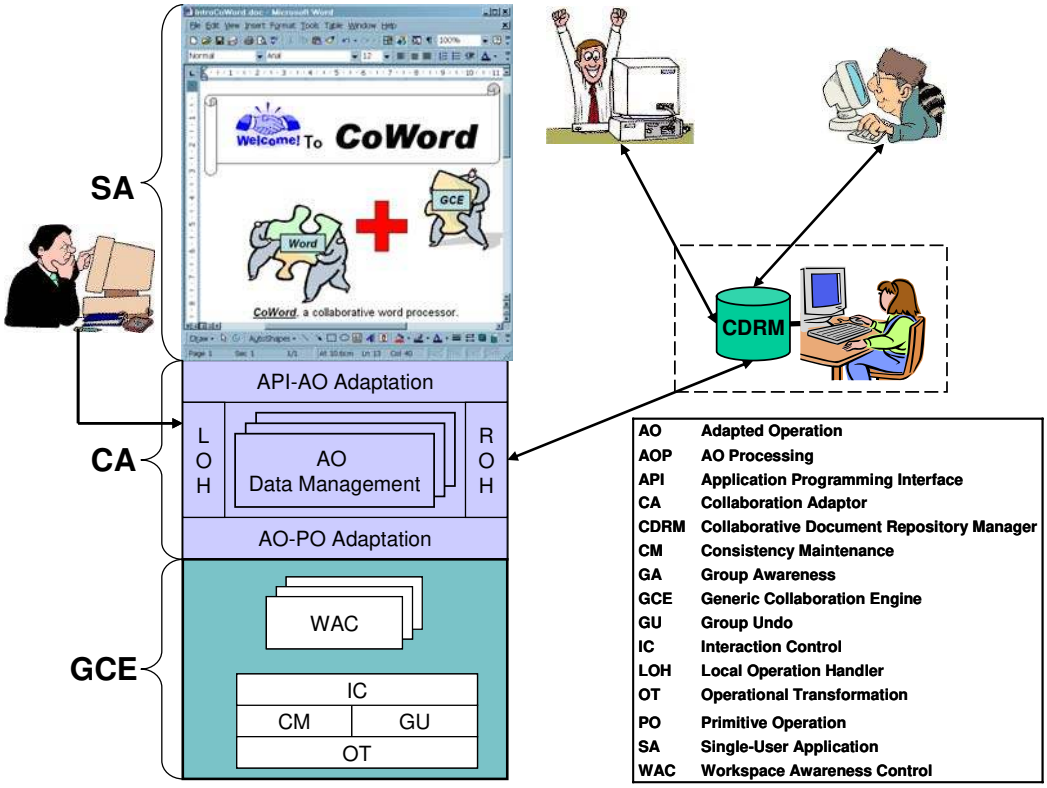


Figure 7.2 The architecture, components and modules of CoWord.

The *Remote Operation Handler* (ROH) is responsible for receiving and processing remote AOs. If the received AO is related to data-manipulation, ROH first uses the AO-PO Adaptation module to translate the AO into suitable POs, which are processed by OT for consistency maintenance, and then ROH calls the API-AO Adaptation module to interpret the transformed AO. For non-data-manipulation AOs, ROH may invoke GCE (via the AO-PO Adaptation module) for some generic service and then, if necessary, calls the API-AO Adaptation

module to interpret the AO. ROH also provides service to propagate local operations to remote sites.

It should be noted that ROH and LOH are implemented as two concurrent threads in CA. Only one of them could be active at any instant of time to ensure the atomicity of local and remote operations. LOH is given a higher priority when both are competing for the control of CA. When ROH is in control of CA in processing a remote AO, the local user interaction with the application is temporarily blocked.

CA also contains several utility modules. One of them is the *AO Data Management* module, which provides services for storing, accessing, and manipulating application-specific data objects contained in AOs. It makes use of the API-AO Adaptation module to manipulate various types of data object transparently.

To illustrate how various modules work together in processing an editing operation, consider the following simple scenario in CoWord. Suppose a user uses the keyboard and/or mouse to create a graphic object in the local Word document, the following will occur at the local site:

(1) The sequence of local input events are intercepted, performed on the local document copy, and translated into an AO *Insert-floatingObj(vp, num, floatingObj)* by LOH.
(2) LOH calls the AO-PO Adaptation module to translate this AO into a PO *Insert(vp, num, objSeq)*, which is then processed (e.g. timestamped) by the OT module in GCE. Moreover, the AO is attached with the same timestamp as its corresponding PO.
(3) LOH uses the service provided by ROH to propagate the timestamped AO to remote sites.

When the AO *Insert-foatingObj(vp, num, floatingObj)* arrives at a remote site, the following will happen:

(1) The AO is received by ROH, which will wait until it gets control over CA.
(2) ROH calls the AO-PO Adaptation module to translate the AO into a PO *Insert(vp, num, objSeq),* which is then processed by the OT module in GCE. Moreover, the transformed PO is used to update the OT-relevant parameters of the AO.
(3) ROH calls the API-AO Adaptation module to interpret the OT-processed AO.

Although CA is application-specific, the CA components of CoWord and CoPowerPoint are both designed in the above architecture. Moreover, they also share many functional modules. This is one of the major reasons that the development effort of the CoPowerPoint system is far less than that for CoWord. This reusability is expected to significantly reduce the effort of developing the CA components of other TA-based systems as well.

7.2.2. The Generic Collaboration Engine

The GCE component provides application-independent collaboration support to the CA component in CoWord and CoPowerPoint. Moreover, GCE can also be reused in other TA-based systems to provide collaboration support.

Operational Transformation, Consistency Maintenance and Group Undo

Operational Transformation (OT) is at the core of GCE for supporting other modules, especially *Consistency Maintenance* (CM) and *Group Undo* (GU). For details of the techniques encapsulated in OT-based CM and GU modules, the reader is referred to Sun et al. (1998), Sun and Ellis (1998), Sun (2000), Sun and Chen (2002), Sun (2002b), Sun et al. (2004) and Sun et al. (2006).

Interaction Control

Based on the OT technique and the replicated system architecture, the *Interaction Control* (IC) module provides support for multiple interaction paradigms/modes, which are characterized by two control parameters: one is Action Control, which

determines who can edit (or act on) the document, and the other is View Control, which determines who can view which part of the document.

The Action Control parameter may take one of the following two values:

(1) Multi-Actor: multiple users are allowed to edit any objects in the document at the same time. This mode is supported by the OT technique.
(2) Single-Actor: a single user is allowed to edit the document at any instant of time. This mode is implemented by a distributed protocol which blocks all but one user's editing operations.

The View Control parameter may take one of the following two values:

(1) Multi-View: multiple users may view different portions of the document, or view any portion of the document in different formats or from different user interface modes (if supported by the original application) at the same time. This mode is naturally supported by the replicated architecture.
(2) Single-View: all users can view the same portion of the document in the same format and from the same user interface mode. This mode is supported by a distributed protocol which blocks all but one user's view changing operations. A single user, who holds the view-floor, is allowed to change the shared-view.

Based on the above interaction control capability, a TA-based system can support a variety of interaction control modes to facilitate different collaboration tasks.

Workspace Awareness Control

The *Workspace Awareness Control* (WAC) module encapsulates an implementation of the MOAF techniques (see Chapter 6) and provides workspace awareness supports to TA-based systems. With the support of this module, developers of TA-based systems can easily implement a variety of existing and new workspace awareness features. Moreover, implementation of a range of

widely-used workspace awareness features (e.g. telepointer, radar view) is also encapsulated. Developers can directly use them in TA-based systems.

7.3. The Prototype System

The CoWord/CoPowerPoint prototype system consists of the following applications:

(1) CDRM Server: a collaboration session and shared document repository manager.
(2) CDRM Client: a client application that provides the user interface to access the session and repository management services provided by CDRM server.
(3) CoWord: a collaborative word processor converted from MS Word.
(4) CoPowerPoint: a collaborative slides authoring and presentation system converted from MS PowerPoint.

7.3.1. CDRM Server and Client

To manage the shared documents and to provide an interface for starting or joining collaborative editing sessions, a Collaborative Document Repository Manager (CDRM) has been designed and implemented based on an Integrated Repository and Session Management (IRSM) technique (Xia et al. 2006).

The CDRM system contains a CDRM server and a client. The CDRM server can be installed on any user's local machine to convert the private document repository (the file system) into a shared document repository to support collaborative editing. In the Internet-based CoWord/CoPowerPoint Demo, however, only one CDRM server is used to provide world-wide users with remote access to the Word and PowerPoint documents stored on a single machine hosted by Griffith University, Brisbane, Australia.

(a)

(b)

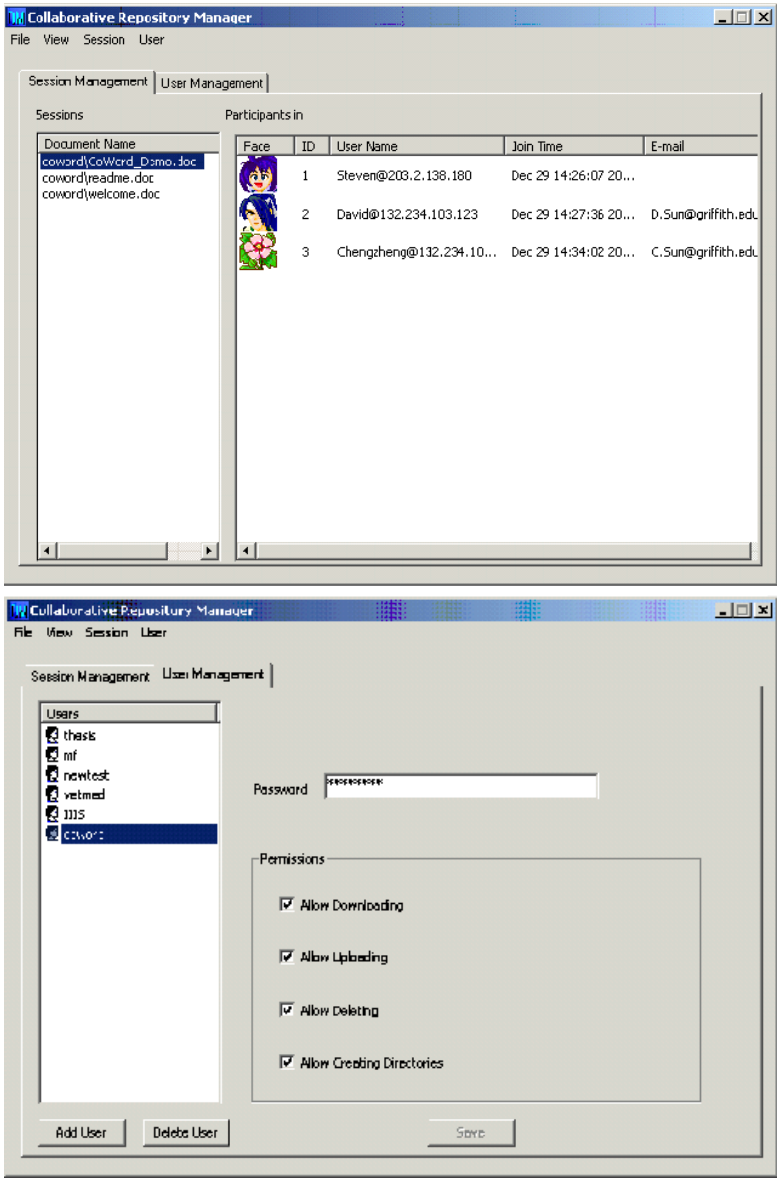


Figure 7.3 The user interface of CDRM server. (a) The Session Management Panel; (b) the User Management Panel.

The user interface of the CDRM server contains a Session Management Panel (Figure 7.3-(a)), from which the user can view detailed information of every ongoing collaborative editing session, and a User Management Panel (Figure 7.3-(b)), from which the user can configure user accounts for accessing sub-repositories.

The CDRM client provides interfaces for users to access services provided by the CDRM server. It is installed in the user's local system, from whose interface (Figure 7.4) the user can perform basic file and folder tasks, such as creating, deleting, copying, and moving files and folders in the shared document repository managed by the CDRM server. Moreover, the remote user can upload and download documents between the local private file system and the shared document repository.

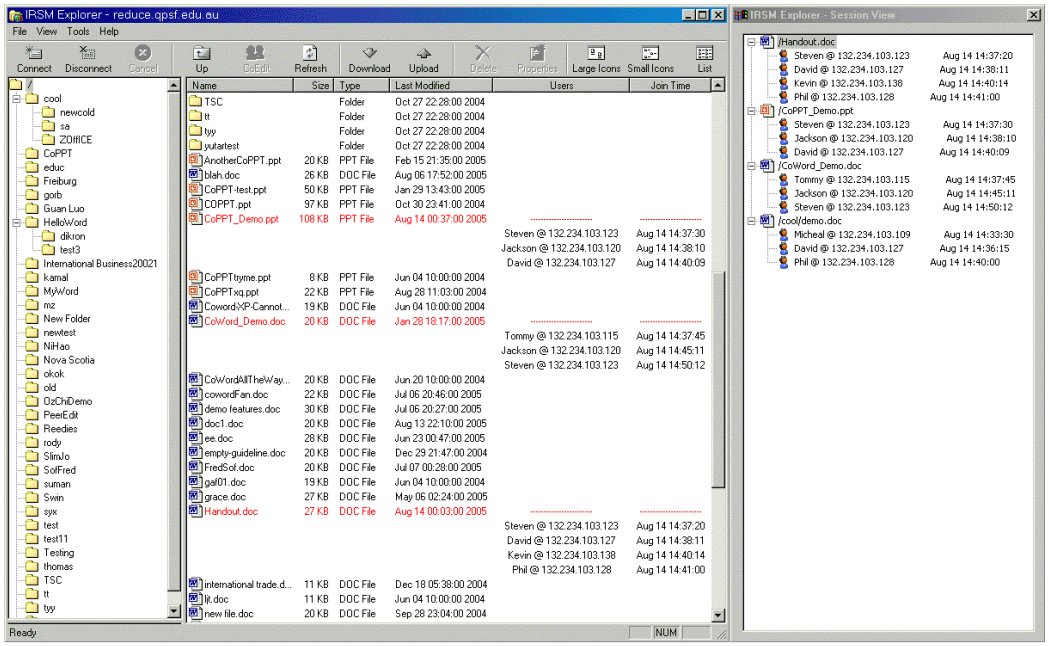


Figure 7.4 The user interface of CDRM client. (a) The Repository View; (b) the Session View.

Based on the IRSM technique, the user can start or join a collaborative editing session of a Word/PowerPoint document in the shared document repository by simply double-clicking the selected document icon from the CDRM client user interface (Figure 7.4-(a)). From the user's point of view, this is no different from starting a normal Word/PowerPoint editing session from the Windows Explorer. However, what happens behind the scenes is quite different.

First, based on the type of the requested document, a suitable collaboration engine, CoWord-Engine or CoPPT-Engine (which is a combination of CA and GCE as discussed in Section 7.1) will be started, with the document path name as the startup parameter. Next the CoWord/CoPowerPoint-Engine sends the document request to the CDRM server. The CDRM server checks whether there is an existing session editing the requested document. If so, it performs a late-joining protocol to join the requesting site into this session. Otherwise, it creates a new session for this requesting site. Next, the CDRM server sends the latest version of the requested document to the requesting site. After receiving the document, the CoWord/CoPPT-Engine starts Word/PowerPoint to edit this document. The session awareness information is also updated in the CDRM client's user interface. After this, the collaborative editing process is under the control of the CoWord/CoPPT-Engine. During a collaboration session, the CoWord/CoPPT-Engine may communicate with the CDRM server to propagate operations or save the edited document back to the shared repository. At the end of a session, the CDRM client will get involved again to clean up the trails of CoWord/CoPPT-Engine on the local machine.

This session management approach has the following advantages. First, it relieves users from the burden of explicit session management actions. The effort required to initiate or join a collaborative editing session is no more than opening a document from the Windows Explorer. Second, it supports the impromptu and flexible collaboration style. Users may join and quit an ongoing session at any time. Third, it solves the common problem of implicit session management approaches, which is the lack of session awareness information. The CDRM client provides users with detailed session awareness information in its user interface. As shown in Figure 7.4-(a), information about users who are currently collaboratively editing documents is listed in the Repository View. With such information, users know clearly whether opening a document will put them into a collaboration session. Moreover, information about each ongoing collaborative editing session is also listed in the Session View (Figure 7.4-(b)). This view

facilitates some pre-planned or formal collaborative activities. The user can easily find the session he/she is interested in and directly join.

7.3.2. CoWord

Collaborative Word Processing

The major objective of CoWord is to convert single-user word-processing functionalities provided by MS Word into collaborative versions. Based on the unconstrained collaboration capabilities provided by GCE and the TA approach presented in this thesis, the current CoWord supports a wide range of collaborative word-processing functionalities, including the following.

(1) Collaborative rich format text editing, with which users can collaboratively insert, delete text and change attributes (e.g. color, size, font type) of text in the shared Word document. Moreover, users can also collaboratively edit attributes of paragraphs (e.g. paragraph alignment, indent, numbering and bulleting).
(2) Collaborative table editing, with which users can collaboratively create, restructure, and fill tables in the shared Word document.
(3) Collaborative graphics editing, with which users can collaboratively create, remove, update (e.g. color, size, position), group and ungroup graphic objects in the shared Word document.
(4) Collaborative document commenting and change tracking, with which users can collaboratively comment on the shared document and edit the document in the change tracking mode. CoWord automatically merges changes from different users.

CoWord allows users to use the above collaboration functionalities without any constraints. For example, while some users are editing the text of the shared document, some others may group graphics objects or edit a table. While some users are editing in the tracking mode, others may be in the normal (non-tracking)

mode. CoWord accommodates all types of concurrent operations and maintains the system consistency.

At the same time, the Word user interface features are preserved. The user can interact with CoWord in the same way he/she interacts with the single-user Word. However, the functionalities triggered by the user's interaction are automatically converted into collaborative versions. For example, when the user clicks the Undo button in the single-user Word, his/her last action is undone, but in CoWord, a collaborative undo function supported by the ANYUNDO algorithm (Sun 2002a) is triggered. Preservation of the user interface features saves users the burden of learning a new system for the purpose of collaboration and thus increases the chance for user acceptance.

Interaction Control

Users may adopt different collaboration styles in collaborative document editing, ranging from single to joint writing styles (Posner and Baecker, 1992). Different interaction control modes are needed to facilitate these collaborative writing styles. For example, to support the impromptu collaborative document writing, an unconstrained collaboration mode is needed, in which any user can edit and view any part of the document. To support the scribed writing mode in which multiple users discuss an issue and one user writes down the discussion result, it is necessary to adopt a Multi-View Single-Actor mode, in which only the scribe can edit the document but discussers are allowed to view any part of the document.

To meet this requirement, CoWord supports the following interaction control modes.

(1) Multi-View and Multi-Actor: multiple users can view and edit any portions of the document at the same time. This mode corresponds to the unconstrained collaboration mode, which is available in collaboration-aware systems, such as REDUCE (Sun et al. 1998) and GRACE (Chen 2001; Sun and Chen 2002).

(2) Multi-View and Single-Actor: multiple users can view any portions of the document, but only a single user can edit the objects in his/her view. This mode is available in some application-sharing systems that support relaxed WYSIWIS, such as the commercial Groove Virtual Office system (Groove Networks Inc. 2006).

(3) Single-View and Multi-Actor: the same portion of the document is viewed by all users, but multiple users can concurrently edit objects in the same view. This mode is, to the best of our knowledge, not available in other existing systems.

(4) Single-View and Single-Actor: the same portion of the document is viewed by all users, and only one user is allowed to edit objects in the shared view. This mode is similar to the strict WYSIWIS and the sequential interaction paradigm supported by generic application-sharing systems.

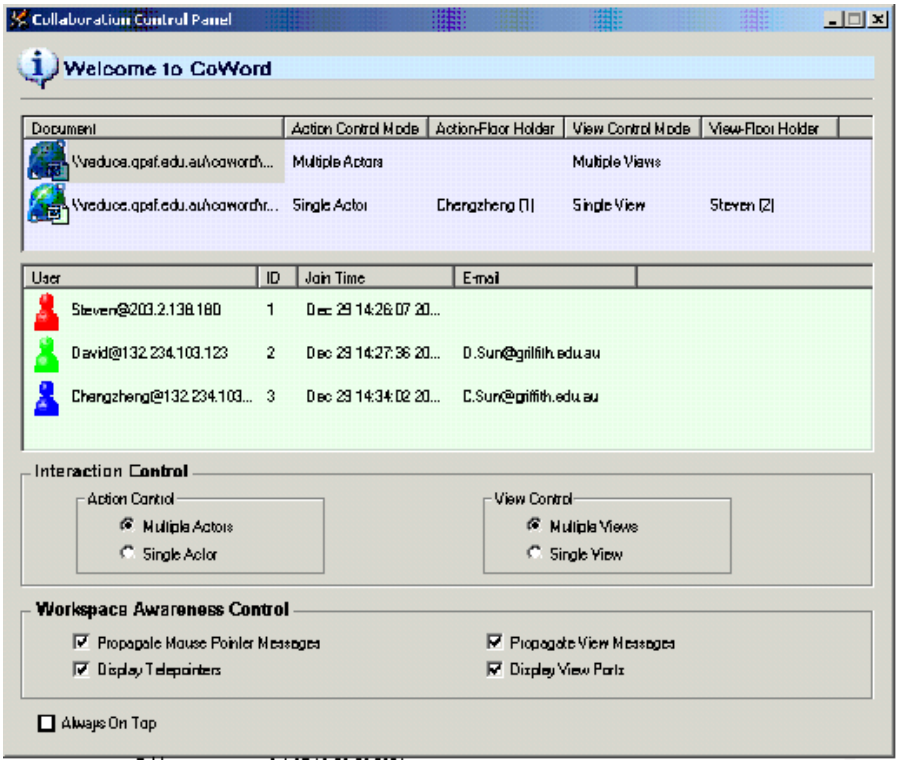


Figure 7.5 The CoWord Control Panel.

The user may initiate an interaction control mode from the CoWord Control Panel (Figure 7.5).

Workspace Awareness Features

Based on the MOAF technique (see Chapter 6), CoWord supports two workspace awareness features, the telepointer and the radar view, as shown in Figure 7.6. From the radar view (on the right side of the workspace), it can be seen that three users (Steven Xia, David Sun and Chengzheng Sun) are viewing an overlapping portion of the document, but one user (David Chen) is viewing a different part of the document. In the workspace, two tele-pointers for David Sun and Chengzheng Sun are displayed since their view ports overlap with Steven's.

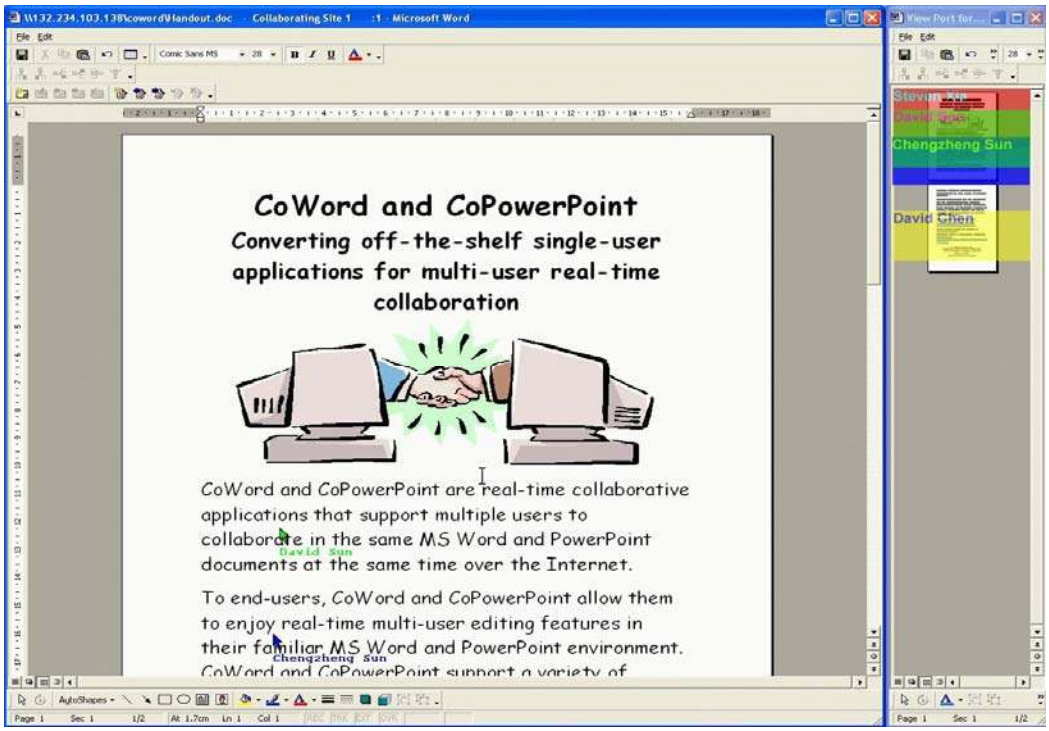


Figure 7.6 Workspace awareness features of CoWord

The user has control over the awareness information display and propagation. The user can enable/disable the display of the telepointers or view ports of other users, and the propagation of his/her own mouse pointer and view port change

messages. These functions are directly supported by the WAC module in GCE. The user can set these options from the CoWord Control Panel (Figure 7.5).

Since a user can be involved in multiple collaborative editing sessions at the same time, the CoWord Control Panel also provides dynamic session information for all ongoing sessions,[11] including the name of the document for each session, the identifiers of current users in each session and their joining times, and the interaction mode associated with each session. As shown in Figure 7.5, the local user is involved in two collaborative sessions. The second session is in a Single-View Single-Actor mode. The action floor and view floor are held by two different users.

7.3.3. CoPowerPoint

MS PowerPoint has functionalities in two categories, which are slides authoring and presentation. CoPowerPoint focuses on converting these two functionality categories into collaborative versions.

Collaborative Slides authoring

A PowerPoint document is organized in multiple levels, including slides, graphic objects and structures inside graphic objects (e.g. the text in a text box). CoPowerPoint supports users to collaboratively edit any objects in the PowerPoint document in any level at any time.

Meanwhile, the user interface features of PowerPoint are preserved while its single-user functionalities are converted into the collaborative version. An interesting outcome of the transparent adaptation of PowerPoint is that CoPowerPoint not only preserves existing single-user PowerPoint interface

[11] The session awareness information displayed by CDRM (Figure 7.3) is similar to that provided by the CoWord Control Panel; the difference is that the former is for all sessions, but the latter is for sessions associated with one particular user.

features, but also creates new multi-user interface features from the combination of multiple single-user interfaces.

Supported by the unconstrained collaboration capability of CoPowerPoint, multiple users are free to choose which interfaces to interact with PowerPoint, which naturally creates new multi-user interface features resulted from the combination of multiple single-user interfaces at the same time.

For example, one user may be in slide-sorter-view, focusing on structuring the overall presentation, while some other users are in the normal view, focusing on creating and updating graphic objects inside individual slides. An interesting interface feature of this combination is that the user in slide-sorter-view can not only freely edit the slide sequence, but also observe the updates made on individual slides by other users in real time. This new feature creates a new usage of an existing interface: the slide-sorter-view interface can be used as a global viewing panel for observing the dynamic contents of all slides. The capability of observing real-time updates on all slides provides a natural group-awareness support to collaborating users. The user in the slide-sorter-view can do a better job in sorting slides thanks to the knowledge of up-to-mini-second updated contents of individual slides. The users in other interfaces (e.g. normal-view) can also take advantage of this group-awareness support by simply running one more PowerPoint instance in slide-sorter-view (on the same machine). In this way, all users in a session can view the global dynamics of the document while working on any parts of the document.

Another interesting combination of multiple single-user PowerPoint interfaces is a collaboration session consisting of one user in the slide-show presentation interface showing the slides to the audience (e.g., using a LCD projector connected to this user's computer), and another one or more users in the slide-view editing interface. In the single-user environment, the contents (including animations) of the document being presented are pre-determined and cannot be revised dynamically. With the combination of slide-show and normal-view

editing interfaces in the same session, it becomes possible to dynamically revise the contents of the document being presented. This new multi-user interface feature can be useful when multiple users are jointly discussing and revising a PowerPoint document at the same time. For example, if an error was found in the document being presented or a revision was suggested by one collaborator, the document can be directly updated from a separate slide-view editing interface and immediately reflected on the slide-show interface, without the need to switch back and forth between the two interfaces. This combination has been an important foundation in supporting collaborative presentation in CoPowerPoint (to be discussed in the next subsection).

There are many other possible combinations of single-user PowerPoint interface features available in unconstrained collaboration sessions. The innovative use and management of these new interface features are interesting topics for future research.

Collaborative Presentation

Computer-supported collaborative presentation applications are an important branch of groupware systems (Isaacs et al. 1994; Gemmel and Bell 1997; Jancke et al. 2000). CoPowerPoint supports collaborative presentation by converting the single-user presentation functionality of PowerPoint into the collaborative version.

Collaborative presentation is a synchronized process in which all participants view the same slide presented by the speaker. Based on the unconstrained collaboration capability, CoPowerPoint supports collaborative presentation with the following synchronization mechanism.

To start a collaborative presentation, all users in the same session enter the slide-show-view. Then all non-speaker users' inputs are blocked. Only the speaker has the privilege to manipulate the slides (including selecting, annotating and editing the current presented slide). The current presented slide chosen by the

speaker is presented on audiences' screens. Annotations made by the speaker are also displayed on audiences' screens.

Moreover, CoPowerPoint also facilitates interaction in collaborative presentation. In many interaction forms (e.g. questioning and discussion) in the presentation, the audience also needs the privilege to manipulate the slides, which is not supported in existing collaborative presentation system. This functional insufficiency often makes interaction in collaborative presentations clumsy (Xia et al. 2005b; 2006b). CoPowerPoint is able to solve this problem based on its unconstrained collaboration capability and other presentation-supporting techniques. Particularly, CoPowerPoint supports the following interaction forms:

(1) Lecturing, in which the speaker delivers the lecture and the audience passively receives it. In this interaction form, the speaker has exclusive control to manipulate the slides. The audience can only view the currently presented slides.
(2) Questioning, in which a questioning audience raises a question and the speaker answers it. While asking a question, the questioning audience is allowed to manipulate the slides. After finishing asking, the speaker takes back the control so that he/she can manipulate the slides while answering the question.
(3) Discussion, in which all participants speak in turn. The speaking user holds exclusive control to manipulate the slides. After a user finishes speaking, control is passed to the next speaking user.
(4) Group discussion, in which users in the same session are divided into groups. In each group, there is a floor circulating among group members, so that they can perform discussion as in (3).

For details about techniques for supporting these interaction forms in CoPowerPoint, the reader is referred to Xia et al. (2005b).

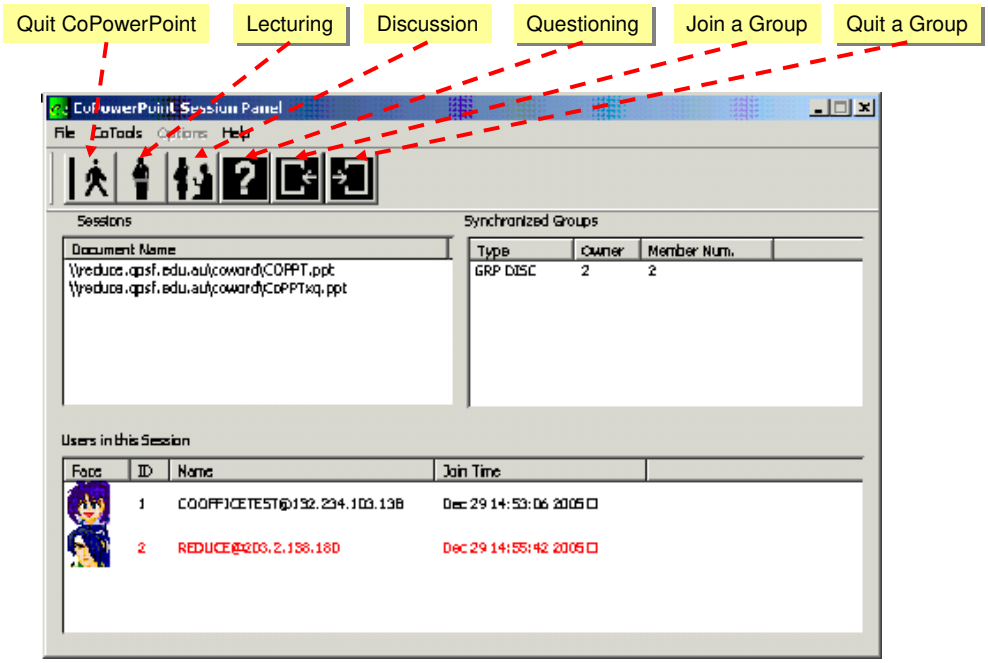


Figure 7.7 The CoPowerPoint Control Panel.

In CoPowerPoint, the user may initiate the above interaction forms, join an existing interaction group and view the current situation of interaction forms from the CoPowerPoint Control Panel, as shown in Figure 7.7.

7.4. Implementation Experiences

CoWord is the first prototype based on the TA approach (in fact, it was the vehicle to drive the development of the TA approach). A group of researchers, collectively with intimate knowledge of the OT technique and its implementation, plus good programming experience with the API of Word and Windows, spent approximately 3 man-years to investigate, design, and implement a publicly demonstrable CoWord prototype. However, the follow-up CoPowerPoint demonstrator was built in less than six man-months. The significant reduction in the CoPowerPoint development time was largely due to the established TA framework and the reuse of software components from CoWord.

In both CoWord and CoPowerPoint, the major development effort was in the collaboration adaptor part. It requires significant effort to accomplish the

adaptation task. On the other hand, the generic collaboration engine was initially converted from the REDUCE engine (Sun 2002a) and evolved into a more generic, independent, and sophisticated component.

The separation of the collaboration adaptor and the generic collaboration engine had allowed us to design, implement and test these two components independently and in parallel, thus accelerating the whole system development. With the availability of the generic collaboration engine component, the adaptation of a new application is reduced into the design and implementation of a new collaboration adaptor for this application.

The CDRM system is an important component of the demonstration system, but this component is independent of the TA approach. It provides session and document repository management services to both CoWord and CoPowerPoint. Moreover, it is able to provide services to other TA-based systems and non-TA based systems. Additional work is needed to transparently integrate this component with existing single-user file managers (e.g. Window Explorer), so that users can use the same file manager to manage both private and shared documents and to launch single-user and multi-user applications.

7.5. Usage Feedback and Experiences

Although no formal usability study has been conducted up to now, considerable usage experiences and feedback have been collected from our research group and thousands of enthusiastic voluntary users around the world. The usage information was collected from two sources. On the one hand, CoWord and CoPowerPoint have been used as the collaboration-supporting tools in our research group. During the over-three-year evolution process of these two systems, we have been using them for collaborative writing of papers, thesis and presentation slides. On the other hand, CoWord and CoPowerPoint were put on the Internet for public demonstration in early 2003, and a free distribution version of these systems was made available in late 2004. Since then, users from different

backgrounds are using these two systems in their different collaborative application environments, and have provided much useful and interesting feedback (and bug reports). The feedback not only helped the improvement of CoWord and CoPowerPoint, but also provided many innovative application cases, which have extended our understanding of the capability of our systems and techniques.

7.5.1. Usage Feedback

Usage feedback so-far has been very encouraging. Users are most happy with the fact that CoWord/CoPowerPoint allows them to use their familiar Word/PowerPoint for collaboration – there is no need to buy or to learn a new tool. Another commonly acknowledged positive point is that CoWord/CoPowerPoint does not impose any specific working style or collaboration process on users, giving users complete freedom in defining their own collaboration processes to meet their divergent and dynamic needs.

Furthermore, users are particularly interested in two collaboration features. The first one is collaborative change-tracking, which is able to mark the authors of changes to the shared document. An example that benefits from this feature is as follows. With an essay collaboratively authored by multiple students, the teacher can clearly identify which student contributed which part. Moreover, it helps the teacher to discover a student's strength and weakness from what he/she contributed to the essay. By reading others' work, students can also learn from each other. Users also appreciate the workspace awareness features, including the telepointer and radar view. Users are excited to see other co-workers' presence and activities in the same collaboration task. These workspace awareness features gave them a strong feeling of involvement in the collaboration.

Users also raised their wishes on new features in our systems for better support of their specific application domains. For example, screenplay writers wish CoWord to support macros that facilitate screenplay formatting; business

document writers wish CoWord to ensure the communication security; and many users wish that other popular single-user applications (e.g. StarOffice (Sun Microsystems 2006b)) should be leveraged to collaborative versions with the TA approach and have similar collaboration capabilities; and the biggest wish of most users has been the availability of a product-quality version of CoWord/CoPowerPoint that could catch up with the newest version of MS Word/PowerPoint. These wishes are important hints for the further development of CoWord/CoPowerPoint and other TA-based systems.

7.5.2. Usage Cases

The majority of users use CoWord/CoPowerPoint for collaborative document creating and editing. However, they collaboratively edit documents in different circumstances. For example, in education circumstances, CoWord/CoPowerPoint has been used for students to collaboratively create slides to study vocabulary words and compose essays as a training of collaboration and communication abilities. In magazine or newspaper publishing circumstances, CoWord has been used to support editors and authors from all over the world to collaboratively edit articles. In screenplay writing circumstances, CoWord has been used to support screenplay writers for different characters to "talk" on the paper in real time.

Collaborative Document Writing

From our usage experience and external users' feedback, the capability of supporting spontaneous collaboration has been identified as an important usability feature. To benefit from using CoWord/CoPowerPoint, users do not have to work at the same time. In fact, even if users do not often work at the same time, they can still benefit from having the same tool to edit the same document at any time.

For example, the editing process of this PhD thesis has been done with CoWord. Three users were involved, including this PhD candidate as the author, and two of his supervisors as the reviewers. One of the supervisors was overseas during the thesis writing process; the other one is on the same campus as the

author is. In this collaborative writing process, the thesis was kept in the collaborative document repository and all users collaboratively worked on it.

The collaboration task was conducted in a spontaneous style. In particular, the author wrote the thesis full time. Reviewers reviewed the thesis and left comments and revisions whenever they had time. With this collaboration style, users often worked at different times, but they also worked at the same time either because they had scheduled a real-time group editing session for online discussion, or because their individual time schedules happened to overlap.[12] Regardless of whether they were working at the same time or at different times, they used the same CoWord tool and had access to the most recent version of the paper at all times; they did not need to distribute versions to each other and to merge multiple versions into one; and they had no worry about inconsistency or incompatible versions.

We consider this anytime collaboration capability as an important usability benefit. An analogy to the telephone technology can be made here: full-duplex telephone technology allows users to talk at anytime, whereas half-duplex technology forces users to take turns in talking. Telephone users often talk in turns – a half-duplex communication protocol, but this half-duplex protocol is best supported by the full-duplex technology. The major usability advantage of full-duplex phones is not only that they allow users to talk at the same time, but also that they allow users to talk at different times without extra effort.

Users have discovered the applicability of CoWord/CoPowerPoint in many interesting application domains other than collaborative document editing. Here are some representative examples.

[12] In the case of unintentional or accidental real-time sessions, co-authors were made aware of each other through the session and workspace awareness support.

Collaborative Court Transcript Creating

One example is the use of CoWord in supporting real-time collaborative creation and use of court room transcripts. The basic setting of the court room application is as follows: one stenographer listens live and types the transcript of the court proceedings; one editor listens to the digital recording and edits the transcript produced by the stenographer;[13] and the judge reads the transcript produced by the editor. The stenographer and the editor are both using MS Word. In the existing court room process, the editor has to wait for the stenographer to finish up the draft transcript before he/she can start editing it (normally one day late); and the judge has to wait for the stenographer and the editor to finish before reading (so the judge needs to take his/her own notes during the court room process).

In several real court room sessions (the longest trial lasted for 5 consecutive days), CoWord was used to allow the stenographer to type and the editor to edit the same transcript at the same time (in a pipeline fashion) so that the final transcript could be produced immediately after each hearing. Moreover, the transcript was presented to the judge via CoWord in real time as well, so the judge could concentrate on analysis and judgement without the burden of taking notes; the judge could directly mark the transcript to highlight or comment on the testimonies which, by his/her judgement, were lies or contradictions, etc.[14] The main benefits here are not only faster creation of better quality court room transcripts but also better support for the judge in the court room process. This is particularly useful and important when a complicated court trail consists of multiple hearings in consecutive days, in which the availability of the previous day's court room transcript is essential for the next day's hearing.

---

[13] The stenographer uses shorthands/abbreviations in typing the transcript in order to keep up with the speed of the court proceedings, so another editor is needed to convert the draft transcript into a formal document.

[14] In CoWord, it is possible to control the propagation of any user's inputs. The judge's highlights/comments on the transcripts could have local effects immediately, but they are not propagated to remote sites until the end of the session, so the stenographer and the editor could not see the judge's comments during the session.

Collaborative Captioning

In contrast to the above example, in which CoWord has significantly improved the performance, CoWord is an essential and foundational tool in another example – collaborative lecture captioning. The purpose of this application is to provide real-time captions about dialogues in lectures for hearing-impaired students, so that they know what the lecturer and students are talking about.

The basic system configuration includes a captioning machine and an editor machine. Moreover, the result caption is projected in a large screen in the classroom. A captioner (i.e. the user of the captioning machine) revoices what was said in the classroom to the speech recognition system running in the captioning machine, which translates the speech into text. The need for revoicing is a result of the technical limitation of the current speech recognition techniques. To achieve a higher accuracy, speech recognition systems can only be trained to recognize the voice of a specific user (i.e. the captioner). Furthermore, the speech recognition system is configured to achieve a high speed at the price of accuracy degradation. The speech recognition system outputs text into a shared document of CoWord[15] with considerable errors. To correct errors, an editor collaboratively edits this document on the editing machine.

Therefore, the captioning application is in fact a collaborative document editing session participated in by multiple users. Moreover, both the captioning and editing are stressful tasks. The captioner and the editor are busy with listening, differentiating speakers and revoicing/editing at the same time. To alleviate the stress and improve the accuracy, multiple captioners and editors are needed to share the workload.

The real-time collaborative editing capability and high responsiveness of CoWord are essential features in this application case. Furthermore, the

[15] The speech recognition system inputs text into CoWord in a simulation of keyboard input, so these inputs can be intercepted by CoWord.

captioning system can be easily extended by allowing each student to have a computer running CoWord to collaboratively view and annotate the local copy of the shared caption document (not propagated to other users). This extension not only allows students to freely browse the caption, but also provides them individual lecture notes with their own annotations.

Due to resource limitation, we have not yet conducted more systematic usability studies on CoWord/CoPowerPoint. With the evolvement of CoWord and CoPowerPoint in functionality and quality, we expect more novel usages will be discovered or invented by users, and more questions about their usability will be answered.

7.6. Summary

This chapter has discussed details of the TA-based prototype systems of this research, which are CoWord, a collaborative word processor converted from MS Word, and CoPowerPoint, a collaborative slides authoring and presentation system converted from MS PowerPoint.

CoWord and CoPowerPoint adopted the same TA-based system architecture. This architecture consists of three components, which are the *Single-user Application* (SA), the *Generic Collaboration Engine* (GCE) and the *Collaboration Adaptor* (CA). The SA component provides conventional single-user functionalities. The GCE component encapsulates application-independent collaboration techniques in the following functional modules: *Operational Transformation* (OT), *Consistency Maintenance* (CM), *Group Undo* (GU), *Interaction Control* (IC) and *Workspace Awareness Control* (WAC). The CA module is responsible for adapting SA to GCE. It contains the following components: *API-AO Adaptation*, *AO-PO Adaptation*, *Local Operation Handler* (LOH), *Remote Operation Handler* (ROH) and *AO Data Management*.

CoWord and CoPowerPoint serve as research platforms in this research. They are used to develop and verify the correctness, feasibility and effectiveness of the TA approach and other collaboration techniques. Moreover, they are also useful groupware systems on their own. They support real-time unconstrained collaborative editing on Word and PowerPoint documents, detailed workspace awareness and flexible session management and flexible interaction control. These two systems also provide reusable components for the development of new TA-based systems. The GCE is application-independent and can be directly reused in other TA-based systems. The TA-based architecture and many functional modules in the CA component can also be reused. The CDRM server and client are able to provide session and document repository management service to any TA-based editing systems.

CoWord and CoPowerPoint have been publicly demonstrated on the web site. Users from different backgrounds have used these two systems in different application circumstances and provided useful feedbacks, including their opinions on the existing collaboration features, their wishes for new features and some interesting usage cases.

Chapter 8

Discussion

As discussed in Chapter 2, replicated generic application-sharing systems have been facing challenging problems in maintaining application consistency, managing access to external resources, and accommodating late-comers. This chapter provides explanations of how and why some of these problems have been simplified or circumvented by CoWord/CoPowerPoint. The applicability of the TA approach to collaboration-transparent and collaboration-aware applications, and its requirements and limitations, are also discussed in this chapter.

8.1. Dealing with Problems Related to the Replicated Architecture

8.1.1. Maintaining Application Consistency

In replicated generic application-sharing systems, the main reason of the difficulty in maintaining system consistency is the absence of application semantic knowledge. Equipped with the application semantic knowledge and the OT technique, CoWord/CoPowerPoint does not require replicas to receive the same inputs from users and other external resources, let alone receiving them in order. Consistency maintenance in the face of concurrency is achieved by means of OT. Consequently, it is not necessary to impose sequential interaction among users for the purpose of ensuring consistency, though sequential interaction (i.e. the Single-Actor mode) can be enforced by the system for the purpose of supporting closely synchronized collaborative work. In the following subsection,

the issues related to the management of non-user external inputs in CoWord/CoPowerPoint are discussed.

8.1.2. Managing Access to External Resources

In CoWord and CoPowerPoint, the problem of managing access to external resources has been significantly simplified by the following factors: (1) the execution of the shared application (i.e. Word or PowerPoint) is mainly driven by the user's interaction; (2) the application-specific collaboration adaptor understands the meaning of the user's interaction; and (3) the primary objective of CoWord/CoPowerPoint is to achieve consistent data-sharing, rather than view-sharing characterized by strict WYSIWIS.

For example, the user may insert the content of an external document (file) into the current in-memory document by interacting with the Word/PowerPoint user interface. Rather than propagating the user's interface activities to remote sites and replaying these interface events in different contexts (which may cause various problems, as identified in Lauwers et al. (1990) and Begole et al. (2001)), CoWord/CoPowerPoint converts the user's interface activities into abstract operations (i.e. an insertion AO in the case of inserting a file). This insertion AO is propagated to remote sites, processed by OT for concurrency control, and finally performed on the remote replica via the application's API, which effectively inserts the file content (represented by the AO's object parameter) into the remote document. The net effect is that the same data content of the file is inserted in all replicas, but not the same view of the interface activities is observed by all users. This is acceptable from data-consistency point of view. Moreover, processing the insertion AOs generated by reading external files is no different from processing insertion AOs generated by the user from the keyboard and mouse.

As another example, in a CoWord session, one or more users can work in the change-tracking mode, and CoWord can automatically track and merge changes

made by all users in real time. In testing this feature, it was noticed that the merged changes (data) are consistent among all replicas, but the hover texts showing the authorship of the changes are different in different replicas. This is because when remote operations from other users are interpreted by the local API, all changes are recorded (by Word) using the local environment variable: user-name. This problem was easily fixed in CoWord since the application-specific adaptor has the knowledge about the authorship of operations, and has the control over the user-name environment variable from the application's API. This problem, however, cannot be resolved by enforcing the consistency of all environment variables (e.g. the same user-name in this case) at all replicas. This is because the semantics of the shared Word/PowerPoint requires the values of the user-name environment variable to be different at different replicas, and operations generated by different users must be recorded under different names.

Moreover, the recorded times for tracked changes may be inconsistent when collaborating users are working in different time zones: the same user's change is recorded at the local site with the local time but with different remote times at remote sites. This is an example where replicated applications get different external inputs from their external clocks. This inconsistency is fixable by the adaptor (since it has all knowledge needed for fixing this problem), but nothing was done about it in the current CoWord version and it was left to the users' interpretation of the time (similar approaches were also used in many Internet applications, such as emails). This example shows the flexibility in CoWord/CoPowerPoint's solution to some externalities, thanks to the knowledge about the nature of the externalities concerned.

Apart from non-user inputs, replicated applications may also generate non-display outputs to external resources, such as files, processes, and network connections, which were found difficult to manage in generic application-sharing environments (Lauwers et al. 1990; Begole et al. 2001). Nevertheless, non-display outputs did not cause special problems in CoWord/CoPowerPoint because all these outputs are initiated by users and can be intercepted and properly processed

by the adaptor. For example, when the user wants to save the document to an external file, he/she must interact with the application, and this interaction will be intercepted by the local adaptor. The adaptor will save the document content back to a shared document repository or to a new location as specified by the user in the interface. Since the adaptor never propagates the user's interaction about file saving to remote replicas, it is guaranteed that the user-initiated file outputs are executed only once.

Word and PowerPoint allow the user to start external applications, such as a web browser, a FTP client from the pull down menu. In CoWord/CoPowerPoint, the user's interactions for launching external applications are not propagated to remote sites, so external applications are only executed at the local site. Some external applications, however, may be launched to update the objects embedded in the current in-memory document, and thus have impact on the consistency of the replicas. For example, the user may start an external photo editor to edit an image object in the document. Since the external photo editor is not under the control of the CoWord/CoPowerPoint adaptor, it is impossible to monitor the user's interactions with the external application. By means of the Windows API, however, it is possible to intercept notification events when the external application has updated the image object or has completed its execution. Upon intercepting such events, suitable *Replacement* AOs (interpreted as a *Delete* and an *Insert*, see Chapter 3) can be generated to represent the net effects of the external application's execution. Processing these *Replacement* AOs is no different from processing other AOs which are generated by the user from the keyboard and mouse in the Word/PowerPoint environment.

The above solutions to external resource management are not generic, but they fit the existing application-specific adaptation framework and do not require additional mechanisms for support. Further investigation is needed to better understand the nature of external resources in various TA-based systems and to devise more general solutions to them.

8.1.3. Accommodating Late-Comers

In CoWord and CoPowerPoint, the problem of accommodating late-comers has been greatly simplified by the fact that the application-specific adaptor has the full knowledge of and access to the application state information needed for initializing a late-comer. A late-comer can be initialized with the current document content and the internal states of the collaboration adaptor and engine of any existing collaborating site. All these states can be packaged, transported, and installed into the late-comer's local CoWord/CoPowerPoint process without the need of migrating a running CoWord/CoPowerPoint process.

To further simplify the initialization of a late-comer, a distributed synchronization protocol has been designed. For details of this protocol and related issues, the reader is referred to Xia et al. (2006).

8.2. Applicability to both Collaboration Awareness and Collaboration Transparency

Collaborative systems have been traditionally classified into two collaboration-transparent systems and collaboration-aware systems (see Chapter 2). The TA approach is applicable to the design of both collaboration-transparent and collaboration-aware systems: the single-user application component in the TA-based system architecture can be a commercial off-the-shelf single-user application (like Word and PowerPoint)), or a newly designed single-user functional component in a collaboration-aware system (like REDUCE (Sun et al. 1998; Sun 2002b)). This new single-user functional component can be designed and implemented in the same way as a stand-alone single-user application without any concerns about collaboration, except that it provides an API suitable for collaboration adaptation.

From early experience of building collaboration-aware systems, we have learnt that not all components in a collaboration-aware system need to be aware of collaboration. For example, in the REDUCE collaborative plain text editor, only the REDUCE engine is aware of collaboration (for consistency maintenance and group undo), but the user editing interface is just a single-user functional component without any knowledge of collaboration. This single-user component has a simple programming interface consisting of two primitive operations *Insert* and *Delete*, which directly match the basic OT technique implemented in the REDUCE engine. Our early experiences with REDUCE had given us the critical insights and inspiration for the development of the TA approach and the design of CoWord and CoPowerPoint systems. As a matter of fact, the first version of CoWord was designed and implemented by replacing the REDUCE single-user interface component with Word, converting the REDUCE engine to a generic and more powerful collaboration engine, and adding a Word-specific adaptor in between.

CoWord/CoPowerPoint can not only support all advanced collaboration features (e.g. high responsiveness, relaxed WYSIWIS, concurrent work, and group undo) which are available in REDUCE, but also support detailed workspace awareness, which is not available in REDUCE. Under the TA approach, the traditional distinction between collaboration-transparent and collaboration-aware applications has blurred: they can be built in the same way and there is no inherent difference between their capabilities in supporting both individual work and group work.

8.3. Suitability for Data-Centric Collaboration

The TA approach is most suitable to building data-centric collaborative applications like CoWord/CoPowerPoint, whose primary objective is to achieve concurrent and consistent data-sharing, rather than strict WYSIWIS view-sharing.

In CoWord/CoPowerPoint, the local user manipulates the shared document via the user interface (the "front-door" of the application), but remote users' operations are integrated into the shared document via the programmer interface (the "back-door" of the application). Consequently, the user can see interface activities generated by him/herself, and see the effects of remote operations initiated by other users, but cannot see all interface activities generated by other users.

Based on consistent data-sharing, flexible view/action control and detailed workspace awareness can be supported in the TA framework. With the support of these techniques, sharing of some aspects of the user interface activities can be achieved. For example, with the support of telepointers and radar views, the user can see the cursor positions of other users if they are viewing an overlapping portion of the document and see the view ports of other users from the radar view. Under the Single-View (Multi/Single-Actor) mode, all users have the same view of the document content, and can see all cursor positions and movements in the same view port. Under no circumstance, however, can the user see the pop-up windows (e.g. dialogue boxes, menus) or hover texts viewable by remote users. The sharing of this kind of remote interface information may be useful for supporting workspace awareness and can be easily implemented in the Single-Actor and Single-View mode. However, the usefulness and implementation complication of supporting this feature in other interaction modes need further investigation.

In our opinion, for data-centric collaborative work like document editing, concurrent and consistent data-sharing is a requirement; flexible sharing of various aspects of the user interface activities (supported by multiple interaction control modes and workspace awareness techniques) is highly desirable and important; strict WYSIWIS view-sharing may be rarely needed. Some other researchers had even strived to achieve consistent data-sharing under heterogeneous user interfaces or even different applications (e.g. DistEdit (Knister and Prakash 1993), and ICT (Li and Li 2002)). Further study is needed to

better understand the requirements for sharing data and views in different collaborative applications and to devise suitable techniques to support them.

8.4. Requirements and Complexities

8.4.1. Basic Requirements to the API

The TA approach requires the single-user application and its execution environment to provide a suitable API (1) which can be used to intercept and replay the user's interactions with the application, and (2) whose data and operation models are adaptable to that of the underlying OT technique.

The first requirement is generally satisfiable by modern single-user interactive applications and their window managers or operating systems. We have found that the second requirement can be met by many members of commercial office software suites (e.g. Microsoft Office (Microsoft Corp. 2006c) and Sun Microsystems StarOffice (Sun Microsystems Inc. 2006b)). Based on the experience from the CoWord and CoPowerPoint work and our initial investigation of other representative single-user applications, we conjectured that these requirements are satisfiable by a wide range of editor-like applications, including various word processors, graphic drawing and design tools, and CAD/CASE systems. Work is on the way to test this conjecture by applying the TA approach, architecture, and the GCE component to new single-user applications from different vendors, in different application domains, and in different platforms.

8.4.2. Complexities of Adaptation Techniques

The data and operation adaptation techniques discussed in this thesis were based on our experience in the CoWord and CoPowerPoint work. They can be used as guidelines and hints in adapting new applications, but they are by no means recipes for solving all problems in adapting new applications. Different applications provide different APIs and hence different ways of addressing

objects; mapping these different object addressing schemes to the generic OT data model requires different strategies and techniques.

For example, data objects in the main body of a Word document are mapped into the OT data model by means of the special *Range* object from the Word API; data objects in one slide of a PowerPoint document are mapped into the OT data model by means of their z-order indices from the PowerPoint API. In these two cases, the mapping is achieved by analyzing, discovering, and using existing features of the API, without additional design and implementation.

However, there are cases in which there is no direct match between the existing API data addressing schemes and the OT data model, and additional work is needed to bridge the gap. One example is the extension of the basic OT data model in order to match the hierarchical addressing schemes in both PowerPoint and Word APIs. There are other cases in which additional work is needed to convert an existing API addressing scheme into the OT data model.

For example, each *Comment* segment in a Word document is mapped into an independent addressing domain in CoWord; but all *Comment* segments in a Word document are actually packed in a single comment store (called Comment Story), which is accessed as a single linear addressing space from the Word API. Consequently, the position of one data object in a *Comment* segment cannot be directly used to address this object in the Comment Story; the offset of this *Comment* in the Comment Story must be used to calculate the correct address.

Many off-the-shelf single-user applications support complex data structures and editing operations. Applying the TA approach on these data and operation models is nontrivial. Advanced adaptation techniques (e.g. the CoTable and CoGroup techniques in Chapter 5) for supporting these data structures and operations need to be designed.

Chapter 9

Conclusions and Future Work

This thesis has contributed an innovative *Transparent Adaptation* (TA) approach and a package of related collaboration techniques for converting single-user applications into multi-user collaborative versions without touching their source code. The research hypothesis is that transparently converted systems can not only have advanced collaboration capabilities that were previously seen only in collaboration-aware systems, but also maintain conventional functionalities and interface features that were previously seen only in commercial off-the-shelf single-user applications. This research has validated this hypothesis by working prototype systems based on the TA approach and related collaboration techniques. This chapter summarizes the main contributions of this research and discusses directions of future work.

9.1. Summary of Contributions

9.1.1. The TA Approach

The most significant contribution of this thesis is the TA approach. TA is an innovative approach to converting existing or new single-user applications into multi-user collaborative versions without changing their source code. It is based on the use of single-user application's API to intercept and replay the user interaction and the use of the OT technique as the underlying collaboration technique. The TA approach contains two aspects: data adaptation and operation adaptation. Data adaptation explores data addressing schemes of the API from the OT point of view, and bridges the data addressing gap between the API and OT.

Operation adaptation bridges the operational gap between the API and OT, and involves the interception, understanding, representation, transformation, and interpretation of user-generated operations.

TA has advanced the state-of-the-art techniques for the development of collaborative systems. TA can be applied to a wide range of off-the-shelf commercial single-user applications. Collaborative systems transparently converted with the TA approach can not only support unconstrained collaboration (e.g. concurrent work and relaxed WYSIWIS), but can also preserve the conventional functionalities and interface features. Collaboration-specific techniques (e.g. workspace awareness and session management techniques) can also be integrated in TA-based systems. These benefits have not been seen in existing collaboration-transparent systems. Moreover, TA is applicable in the development of both collaboration-transparent and collaboration-aware systems.

Another contribution of this work is the TA-based collaborative system architecture consisting of three components: (1) the single-user application, which provides conventional single-user functionalities and interface features; (2) the collaboration adaptor, which provides application-specific collaboration capabilities; and (3) the generic collaboration engine, which provides application-independent collaboration capabilities. The separation of single-user functionalities from multi-user collaboration capabilities, and the separation of application-specific collaboration capabilities from generic collaboration capabilities in this three-layer system architecture help to reduce the complexity of collaborative system design, and increase the modularity and reusability of collaborative system components.

9.1.2. Extensions to the OT Technique

This thesis work discovered that the applicability of OT is dependent on the addressing relationship among data objects in the shared document (accessed from the API), but independent of the visual relationship among data objects

(presented at the user interface), or the internal relationship among data objects (defined by their class definitions). This discovery is significant because it not only paved the way to apply OT to Word and PowerPoint, but also advanced our understanding of the nature of OT, which may inspire new explorations and applications of OT.

Moreover, this thesis has made important technical contributions to OT with two extensions. The first contribution is the extension of the OT data model from a single linear addressing space to XOTDM (eXtended OT Data Model): a tree of multiple linear addressing domains, together with the (*n, p*) vector-based addressing scheme and transformation functions. Another contribution is the extension of OT, from supporting two primitive operations *Insert* and *Delete,* to supporting arbitrary complex application operations. This extension consists of two parts: one is a generic extension of the OT operation model to include a new primitive operation *Update*; and the other is an application-specific OT extension, which translates application-specific operations into generic primitive operations for OT processing.

The basic OT technique was only able to support collaborative insertion and deletion of plain text character. These extensions have leveraged it to a generic collaboration technique which can be used to support unconstrained collaboration on data structures and editing functionalities with complex semantics.

9.1.3. Advanced Adaptation Techniques for Complex Application Semantics

This thesis work has contributed a package of advanced adaptation techniques for complex data and operation models, including CoTable and CoGroup.

CoTable and CoGroup have extended the capability and applicability of the basic TA approach, which is able to adapt common data and operation models (e.g. rich format text editing and graphics editing). With the support of CoTable and CoGroup, TA-based systems can support unconstrained collaboration on

tables and graphic object grouping. Meanwhile, the applicability of the underlying OT technique is also extended. With these techniques, OT can provide support for collaborative table editing and graphic object grouping. Moreover, CoTable and CoGroup have also enriched the knowledge of both TA and OT. CoTable provides guidelines and demonstrations for adapting complex data models and editing operations defined on these data models. CoGroup demonstrates techniques for resolving AO-level conflicts and achieving desirable effects with an extension to OT and the interaction between OT and adaptation layers. Methodologies and ideas in these techniques can be reused to design adaptation techniques for other complex data and operation models.

9.1.4. TA-Based Workspace Awareness Technique

This thesis has also contributed a TA-based framework for supporting workspace awareness called MOAF. Workspace awareness techniques have been well developed and applied in collaboration-aware systems, but are difficult to be applied in collaboration-transparent systems. Applying workspace awareness in TA-based systems not only improves the usability, but also confirms the research hypothesis. Furthermore, this technique has shown its innovative capabilities, not seen in existing techniques.

The MOAF technique is able to accommodate different object association and graphics representation requirements of workspace awareness features, so can be used to support a wide range of workspace awareness features (e.g. telepointer, radar view, etc). Moreover, it can be easily extended to support new workspace awareness features. Most importantly, based on the capabilities of the underlying TA and OT, MOAF-supported workspace awareness features are able to deliver correct and precise workspace awareness information in the face of dynamic and concurrent content and view changes.

9.1.5. Experimental Prototype Systems

Another contribution of this research is the construction of two experimental prototype systems – CoWord and CoPowerPoint. These two systems are transparently adapted from off-the-shelf commercial single-user applications – MS Word and PowerPoint. CoWord and CoPowerPoint not only support unconstrained collaborative editing on Word and PowerPoint documents, they also preserve the conventional functionalities and interface features of Word and PowerPoint. Moreover, CoWord and CoPowerPoint have integrated a package of collaboration-specific techniques, including detailed workspace awareness, flexible session management, interaction control and interactive presentation.

CoWord and CoPowerPoint have achieved their major goals, which are to verify the correctness and generality of the TA approach and to act as benches for the development of other collaboration techniques. Meanwhile, they are also useful collaborative editing systems on their own. These two systems have been publicly demonstrated in major conferences and our web site. Voluntary users from all over the world have used them in different application circumstances. Moreover, the generic collaborative engine component can be directly reused in building other TA-based systems. Many functional modules of CoWord and CoPowerPoint collaboration adaptor can also be reused.

9.2. Future Work

This research has pioneered a new approach to building groupware systems. At the same time, it raises a number of research issues worth exploring in the future.

The major future work is to extend the TA approach to more mainstream commercial single-user applications, including web design systems, CAD and CASE systems. We anticipate many challenges and opportunities ahead of us as TA is applied to more and more new applications. Some interesting topics are recommended below for future exploration:

(1) **Extensions to TA**. The current TA framework and advanced adaptation techniques are able to convert a range of frequently used functionalities by adapting their data and operation models. While other single-user applications may have different data and operation models, new adaptation techniques need to be designed. With these new techniques, the generality and applicability of TA and the underlying OT will be significantly extended.

(2) **Exploring new collaboration features**. New target single-user applications have different functionalities and interface features from Word and PowerPoint. When these functionalities and interface features are converted into collaborative versions, innovative collaboration features may be generated. The usefulness and management techniques of these features are interesting research topics.

(3) **Incorporation and improvement of collaboration-specific techniques**. In this research, a package of collaboration-specific techniques (e.g. workspace awareness and session management techniques) have been extended and incorporated into CoWord and CoPowerPoint. New TA-based systems may have requirements on other collaboration-specific techniques, such as flexible notification (Shen and Sun 2002; Patterson et al. 1996) and fine-grain optional locking (Sun 2002b). Incorporating these techniques into TA-based systems and applying them in novel collaboration tasks may raise requirements for improving these techniques. Moreover, new TA-based systems can also be used as benches to verify the usefulness and effectiveness of these techniques.

The usability study on TA-based systems is also an important future work. In the near future, formal usability studies will be conducted on CoWord and CoPowerPoint. These studies may cover a range of issues, including how easily users could learn and use TA-based systems, which collaboration tasks users use TA-based system to perform, which collaboration features are more helpful and which are less, and which new features users hope to have. TA-based systems integrate many collaboration techniques, so some usability issues are also generally applicable to other groupware systems. Information collected in these

studies can help improve both TA-related techniques and other collaboration techniques.

References

Abdel-Wahab, H. and Peit, M. 1991. "XTV: A framework for sharing x window clients in remote synchronous collaboration". In *Proc. IEEE Tricomm*. pp. 159 – 167, April 1991.

Ahuja, S. R., Ensor, J. R., and Lucco, S. E. 1990. "A comparison of application sharing mechanisms in real-time desktop conferencing systems". In *Proc. the Conference on office information Systems*, pp. 238 – 248, April 1990.

Basili, V. R., Briand, L. C., and Melo, W. L. 1996. "How reuse influences productivity in object-oriented systems". *Communication of ACM,* 39(10), pp. 104 – 116, October 1996.

Baecker, R. 1992. "Groupware and computer-supported cooperative work". Morgan Kaufmann, 1992.

Baecker, R., Nastos, D., Posner, I., and Mawry, K. 1993. "The user-centered iterative design of collaborative writing software". In *Proc. ACM InterCHI"93 Conference on Human Factors in Computing Systems,* pp. 399 – 405, April 1993.

Beaudouin-Lafon, M. and Karsenty, A. 1992. "Transparency and awareness in a real-time groupware system". In *Proc. the 5th Annual ACM Symposium on User interface Software and Technology*, pp. 171 – 180, November 1992.

Begole, J., Smith, R. B., Struble, C. A., and Shaffer, C. A. 2001. "Resource sharing for replicated synchronous groupware". *IEEE/ACM Trans. on Network,* 9(6), pp. 833 – 843, December 2001.

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

Bharat, K. A. and Cardelli, L. 1995. “Migratory applications”. In *Proc. the Eight Annual ACM Symposium on User Interface Software Technology*, pp, 133 – 142, December 1995.

Bly, S. A., Harrison, S. R., and Irwin, S. 1993. “Media spaces: bringing people together in a video, audio, and computing environment”. *Communication of ACM*, 36(1), pp. 28 – 46, January 1993.

Chen, D. and Sun, C. 1999. “A distributed algorithm for graphic objects replication in real-time group editors”. In *Proc. the International ACM SIGGROUP Conference on Supporting Group Work*, pp. 121 – 130, November 1999.

Chen, D. and Sun, C. 2001. “Optional instance locking in distributed collaborative graphics editing systems.” In *Proc. the Eighth International Conference on Parallel and Distributed Systems*, pp. 109 – 116, June 2001.

Chen, D. 2001. “Consistency maintenance in collaborative graphics editing systems”, *PhD thesis*, Griffith University, Australia, 2001.

Chung, G. and Dewan, P. 1996. “A mechanism for supporting client migration in a shared window system”. In *Proc. of ACM Symposium on User Interface Software and Technology*, pp. 11 – 20, November 1996.

Chung, G., Jeffery, K., and Abdel-Wahab, H. 1993. “Accommodating late comers in shared window systems”. *IEEE Computer*, 26(1), pp. 72 – 74, January 1993.

CoWord Demo 2006. CoWord Demonstration Centre, http://reduce.qpsf.edu.au/coword/, last accessed: February 2006.

CoPowerPoint Demo 2006. CoPowerPoint Demonstration Centre, http://reduce.qpsf.edu.au/copowerpoint/, last accessed: February 2006.

Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. 1990. “MMConf: an infrastructure for building shared multimedia applications”. In

Proc. the ACM Conference on Computer-Supported Cooperative Work, New York, pp. 329 – 342, November 1990.

Davis, A., Sun, C., and J. Lu. 2002. "Generalizing operational transformation to the standard general markup language", In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 58 – 67, November 2002.

Douglis, F. 1990. "Transparent Process Migration in the Sprite Operating System". *PhD. Thesis, Technical Report UCB/CSD 90/598, CSD (EECS),* University of California, Berkeley, 1990.

Dourish, P. 1996. "Consistency guarantees: exploiting application semantics for consistency management in a collaboration toolkit". In *Proc. the ACM Conf. on Computer-Supported Cooperative Work*, pp. 268 – 277, November 1996.

Dyck, J., Gutwin, C., Subramanian, S., and Fedak, C. 2004. "High-performance telepointers". In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 172 – 181, November 2004.

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

Ellis, C. A., and Gibbs, S. J. 1989. "Concurrency control in groupware systems", In *Proc. ACM Conference on Management of Data*, pp. 399 – 407, 1989.

Ellis, C. A., S. J. Gibbs and G. L. Rein. 1991. "Groupware: some issues and experiences". *Communications of the ACM,* 34(1): pp. 38 – 59, January 1991.

Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E., Pier, K., Tang, J., and Welch, B. 1992. "Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration". In *Proc. the SIGCHI Conference on Human Factors in Computing Systems*, pp. 599 – 607, June 1992.

Engelbart, D. and English, W. 1968. “A research center for augmenting human intellect”. In *Proc. the Fall Joint Computing Conference*, vol. 33, pp. 395 – 410, December 1968.

Engelbart, D. 1975. “NLS teleconferencing features: The journal, and shared-screen telephoning”. In *Proc. the IEEE Fall COMPCON. IEEE*, 173 – 176, July 1975.

Forsdick, H. 1985. “Explorations in real-time multimedia conferencing”. In *Proc. the 2nd International Symposium on Computer Message Systems,* pp. 299 – 315, September 1985.

Fuller, D. A., Mujica, S. T., and Pino, J. A. 1993. “The design of an object-oriented collaborative spreadsheet with version control and history management”. In *Proc. the ACM/SIGAPP Symposium on Applied Computing: States of the Art and Practice*, pp. 416 – 423, February 1993.

Garfinkel, D., Welti, B., and Yip, T. 1994. “HP SharedX: A tool for real-time collaboration”. *HP Journal 45(2),* pp. 23 – 36, April 1994.

Gemmel, D.J. and Bell, C.G. 1997. “Noncollaborative telepresentation comes to the age”. *Communication of the ACM*, 40(4), pp. 79 – 89, 1997.

Greenberg, S. 1990. “Sharing views and interactions with single-user applications”. In *Proc. the Conference on office information Systems*, pp. 227 – 237, April 1990.

Greenberg, S. 1991. “Personalizable groupware: accommodating individual roles and group differences”. In *Proc. the European Conference on Computer-Supported Cooperative Work (ECSCW),* September 1991.

Greenberg, S., Gutwin, C., and Roseman, M. 1996. “Semantic telepointers for groupwares”. In *Proc. Australian Conference on Computer-Human Interaction*, pp. 54 – 61, November 1996.

Greenberg, S., Roseman, M., Webster, D., and Bohnet, R. 1992. “Issues and experiences designing and implementing two group drawing tools”. In *Proc. Hawaii International Conference on Systems Sciences*, pp. 138 – 150, January 1992.

Greenberg, S., and Roseman, M. 1996. “Groupweb, a www browser as real time groupware”. In *Proc*. *Conference companion on Human factors in computing systems: common ground*, pp. 271 – 272, April 1996.

Greenhalgh, C. and Benford, S. 1995. “MASSIVE: a collaborative virtual environment for teleconferencing”. *ACM Transaction on Computer-Human Interaction*, (2)3, pp. 239 – 261, September 1995.

Groove Networks Inc. 20006. Groove Virtual Office. http://www.groove.net/index.cfm/pagename/VirtualOffice/, last accessed February 2006.

Grudin, J. 1994a. “Computer-supported cooperative work: Its history and participation”. *IEEE Computer,* 27(5): 19 – 26.

Grudin, C. 1994b. “Eight challenges for groupware developers”. *Communications of the ACM*, 37(1), pp. 92–105, January 1994.

Gutwin, C., Greenberg, S., and Roseman, M. 1996. “Workspace awareness support with radar views”. In *Proc. Conference on Human Factors in Computing Systems,* pp. 13 – 18, April 1996.

Gutwin, C. 1997. “Workspace awareness in real-time distributed groupware”. *PhD thesis*, University of Calgary, Calgary, Canada, 1997.

Gutwin, C., Dyck, J., and Burkitt, J. 2003. “Using cursor prediction to smooth telepointer jitter”. In *Proc. ACM SIGGROUP Conference on Supporting Group work*, pp. 294 – 301, November 2003.

Gutwin, C. and Greenberg, S. 1996. “Workspace Awareness for Groupware”. In *Proc. the Conference on Human Factors in Computing Systems,* pp. 208 – 209, April 1996.

Gutwin, C., Greenberg, S., Blum, R., and Dyck, J. 2005. “Supporting informal collaboration in shared-workspace groupware”. *HCI Technical Report 2005 – 01,* the Interaction Lab, University of Saskatchewan, Canada*,* 2005.

Gutwin, C. and Greenberg, S. 1999. “The effects of workspace awareness support on the usability of real-time distributed groupware”. *ACM Transaction on Computer-Human Interaction*, 6(3), pp. 243 – 281, 1999.

Gutwin, C. and Greenberg, S. 2002. “A descriptive framework of workspace awareness for real-time groupware”. In *Proc. ACM Conference on Computer Supported Cooperative Work*, 11, 3, pp. 411 – 446, July 2002.

Gutwin, C., Greenberg, S. and Roseman, M. 1996a. “Workspace awareness in real-time distributed groupware: framework, widgets, and evaluation”. In *Proc. ACM Conference on Computer-Human Interaction, Springer-Verlag*, pp. 281 – 298, 1996.

Gutwin, C., Greenberg, S., and Roseman, M. 1996b. “Workspace awareness support with radar views”. In *Proc. Conference on Human Factors in Computing Systems*, pp. 13 – 18, April 1996.

Gutwin, C., and Penner, R. 2002. “Improving interpretation of remote gestures with telepointer traces”. In *Proc. ACM Conference on Computer-Supported Cooperative Work*, pp. 49 – 57, November 2002.

Hill, J. and Gutwin, C. 2003. “Awareness support in a groupware widget toolkit”. In *Proc. the 2003 International ACM SIGGROUP Conference on Supporting Group Work,* pp. 258 – 267, November 2003.

Ignat, C. and Norrie, M. 2003. “Customizable collaborative editor relying on treeOPT algorithm”. In *Proc. the European Conference of Computer-Supported Cooperative Work*, pp. 315 – 324, September 2003.

Ignat, C. and Norrie, M. C. 2004. “Grouping in collaborative graphical editors”. In *Proc. of Computer Supported Cooperative Work*. pp. 447 – 456, November 2004.

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

Isaacs, E.A., Morris, T. and Rodriguez, T.K. 1994. “A forum for supporting interactive presentations to distributed audiences”, In *Proc. ACM Conference on Computer-Supported Cooperative Work*, pp. 405 – 416, October 1994.

Iseminger, D. 2000. “Automation. COM+ Developer’s Reference Library”, vol. 4. Redmond: Microsoft Press.

Jancke, G., Grudin, J., and Gupta. A. 2000. “Presenting to local and remote audiences: design and use of the TELEP system”, In *Proc. the SIGCHI Conference on Human Factors in Computing Systems*, pp. 384 – 391, April 2000.

Kanawati, R. 1997. “LICRA: A replicated-data management algorithm for distributed synchronous groupware application”. *Parallel Computing*, 22, pp. 1733–1746, 1997.

Karsenty, A. and Beaudouin-Lafon, M. 1993. “An algorithm for distributed groupware applications”. In *Prof. 13th Conference on Distributed Groupware Computing Systems*, pp. 195 – 202, May 1993.

Karsenty, A., Tronche, C., and Beaudouin-Lafon, M. 1993. “Groupdesign: shared editing in a heterogeneous environment”, *Usenix Journal of Computing Systems,* 6(2), pp. 167 – 195, 1993.

Knister, M. J. and Prakash, A. 1990. "DistEdit: a distributed toolkit for supporting multiple group editors". In *Proc. ACM Conference on Computer Supported Cooperative Work*, pp. 343 – 355, October 1990.

Lamport, L. 1978. "Time, clocks, and the ordering of events in a distributed system". *Communication of ACM,* 21(7), pp. 558 – 565, July 1978.

Lantz, K. 1986. "An experiment in integrating multimedia conferencing". In *Proc. the ACM Conference on Computer-Supported Cooperative Work*. pp. 267 – 275, December 1986.

Lauwers, J., Joseph, T. A., Lantz, K., and Romanow, A. L. 1990. "Replicated architectures for shared window systems: A critique". In *Proc. the ACM Conference on Organization Information Systems*. pp. 249 – 260, March 1990.

Lauwers J. C. and Lantz. K. A. 1990. "Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems". In *Proc. ACM CHI"90 Conference on Human Factors in Computing Systems*, pp. 303 – 311, April 1990.

Leland, M. D. P., Fish, R. S., and Kraut, R. E. 1988. "Collaborative document production using Quilt". In *Proc. the Conference on Computer-Supported Cooperative Work*, pp. 206 – 215, September 1988.

Li, D. and Li, R. 2002. "Transparent sharing and interoperation of heterogeneous single-user applications". In *Proc. the ACM Conference on Computer-Supported Cooperative Work,* pp. 246 – 255, November 2002.

Li, D. and Li, R. 2004. "Preserving operation effects relation in group editors". In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 457 – 466, November 2004.

Li, R. and Li, D. 2005a. "A landmark-based transformation approach to concurrency control in group editors". In *Proc. the ACM SIGGROUP Conference on Supporting Group Work*, pp. 284 – 293*,* November 2005.

Li, R. and Li, D. 2005b. “Commutativity-Based Concurrency Control in Groupware”. In *Proc. The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, December 2005.

Mehlenbacher, B., Hardin, B., Barrett, C., and Clagett, J. 1994. “Multi-user domains and virtual campuses: implications for computer-mediated collaboration and technical communication”. In *Proc. the 12th Annual international Conference on Systems Documentation: Technical Communications At the Great Divide*, pp. 213 – 219, October 1994.

Microsoft Corp. 2006a. Microsoft NetMeeting. http://www.microsoft.com/netmeeting/, last accessed: February 2006.

Microsoft Corp. 2006b. Microsoft Word Objects. http://msdn.microsoft.com/library/en-us/off2000/html /wotocobjectmodelapplication.asp, last accessed: February 2006.

Microsoft Corp. 2006c. Microsoft Office. http://office.microsoft.com/, last accessed February 2006.

Milojičić, D., Zint, W., Dangel, A., and Giese, P.1993. “Task Migration on the top of the Mach Microkernel”. In *Proc the third USENIX Mach Symposium*, pp. 273 – 290, April 1993.

Nardi, B. A. and Miller, J. R. 1990. “An ethnographic study of distributed problem solving in spreadsheet development”, In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 197 – 208, October 1990.

NCSA 2005. NCSA Collage. http://archive.ncsa.uiuc.edu/SDG /Software/XCollage/collage.html, last accessed October 2005.

Netopia Inc. 2006. Timbuktu. http://www.netopia.com/software/products/tb2/, last accessed February 2006.

Newman-Wolfe, R. E., Webb, M. L., and Montes, M. 1992. “Implicit locking in the Ensemble concurrent object-oriented graphics editor”. In *Proc. ACM*

Conference on Computer-Supported Cooperative Work, pp. 265 – 272, November 1992.

Nguyen, D. and Canny, J. 2005. "MultiView: spatially faithful group video conferencing". In *Proc. the SIGCHI Conference on Human Factors in Computing Systems*. pp. 799 – 808, April 2005.

Palmer, C., and Cormak, G. 1998. "Operation transforms for a distributed shared spreadsheet". In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 69 – 78, November 1998.

Poltrock, S. and Grudin, J. 1994. "Computer supported cooperative work and groupware". In *Proc. Conference Companion on Human Factors in Computing Systems*, pp.355 – 356, April 1994.

Patterson, J., Day, M., and Kucan, J. 1996. "Notification servers for synchronous groupware". In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 122 – 129, November 1996.

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

Posner, I.R. and Baecker, R.M. 1992. "How people write together". In *Proc. 25th Hawaii International Conference on System Sciences,* pp. 127 – 138, 1992.

RealVNC 2006. RealVNC VNC, http://www.realvnc.com/, last accessed: February 2006.

Ressel, M., Nitsche-Ruhland, D., and Gunzenhauser, R. 1996. "An integrating, transformation-oriented approach to concurrency control and undo in group editors". In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 288 – 297, November 1996.

Rodham, K. J., and Olsen D. R. 1994. “Smart telepointers: maintaining telepointer consistency in the presence of user interface customization”. *ACM Transactions on Graphics*, 13(3), pp. 300 – 307, July 1994.

Roseman, M. and Greenberg, S. 1992. “GROUPKIT: a groupware toolkit for building real-time conferencing applications”. In *Proc. the 1992 ACM Conference on Computer-Supported Cooperative Work,* pp. 43 – 50, December 1992.

Roseman, M., and Greenberg, S. 1996a. “Building real-time groupware with groupkit, a groupware toolkit”. *ACM Transactions on Computer-Human Interaction*, 3(1), pp. 66 – 106, 1996.

Roseman, M. and Greenberg, S. 1996b. “TeamRooms: network places for collaboration”. In *Proc. of the 1996 ACM Conference on Computer Supported Cooperative Work*, pp. 325 – 333, March 1996.

Shen, H. and Sun, C. 2002. “A flexible notification framework for collaborative systems”. In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 77 – 86, November 2002.

Shen, H. and Sun, C. 2004. “Improving real-time collaboration with highlighting”. *Future Generation Computer Systems,* 20(4), May, 2004.

Silberhorn, H. 2001. “TabulaMagica – an integrated approach to manage complex tables”, In *Proc. the ACM Symposium on Document Engineering*, pp. 68 – 75, November 2001.

Stefik, M., Bobrow, D., Foster, G., Lanning, S., and Tatar, D. 1987. “WYSIWIS revised: early experiences with multiuser interfaces”. *ACM Transactions on Office Information System*, 5(2), pp.147 – 167, April 1987.

Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., and Suchman, L. 1987. “Beyond the chalkboard: computer support for collaboration and problem solving in meetings”. *Communication of the ACM*, 30(1), pp. 32 – 47, January 1987.

Suleiman M., Cart M., and Ferrié J. 1997. “Serialization of concurrent operations in distributed collaborative environment”. In Proc. the ACM International Conference on Supporting Group Work, pp. 435 – 445, November 1997.

Suleiman, M., Cart, M., and Ferrie, J. 1998. “Concurrent operations in a distributed and mobile collaborative environment”. In *Proc. the IEEE Fourteenth International Conf. on Data Engineering*, pp 36 – 45, February 1998.

Sun, C. 2002a. “Undo as concurrent inverse in group editors”, *ACM Transactions on Computer-Human Interaction*, 9(4), pp. 309 – 361, December 2002.

Sun, C. 2002b. “Optional and responsive fine-grain locking in Internet-based collaborative systems”. *IEEE Transactions on Parallel and Distributed Systems*, 13(9), pp. 994 – 1008, September 2002.

Sun, C. and Chen, D. 2002. “Consistency maintenance in real-time collaborative graphics editing systems”. *ACM Transactions on Computer-Human Interaction*, 9(1), pp. 1 – 41, May 2002.

Sun, C. and Ellis, C. A. 1998. “Operational transformation in real-time group editors: issues, algorithms, and achievements”. In *Proc. the ACM Conference on Computer-Supported Cooperative Work*, pp. 59 – 68, November 1998.

Sun, C., Jia, X., Zhang, Y. and Chen, D. 1998. “Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems”. *ACM Transaction on Computer-Human Interaction*, 5(1): pp. 63 – 108, March 1998.

Sun, C., Yang, Y., Zhang, Y., and Chen, D. 1996. “A consistency model and supporting schemes for real-time cooperative editing systems”. In *Proc. the 19th Australian Computer Science Conference*, pp. 582–591, January 1996.

Sun, C., Xia, S., Sun, D., Chen, D., Shen, H. and Cai, W. 2006. “Transparent adaptation of single-user applications for multi-user real-time collaboration”. To appear in *ACM Transactions in Human-Computer Interaction*.

Sun, D., and Sun, C. 2006. “Operation context and context-based operational transformation”. To appear in *the ACM Conference on Computer-Supported Cooperative Work*, November 2006.

Sun, D., Xia, S., Sun, C., and Chen, D. 2004. “Operational transformation for collaborative word processing”, In *Proc. ACM Conference on Computer Supported Cooperative Work*, pp. 437 – 446, November 2004.

Sun Microsystems Inc. 2006a. SunForum. http://www.sun.com/desktop/products/software/sunforum, last accessed: February 2006.

Sun Microsystems Inc. 2006b. StarOffice. http://wwws.sun.com/software/star/staroffice, last accessed: February 2006.

Tang, J. C., and Minneman, S. L. 1990. “Videodraw: A video interface for collaborative drawing”. In *Proc. of ACM SIGCHI Conference on Human Factors in Computing Systems*, pp. 313 – 320, April 1990.

Tang, J. C., and Minneman, S. L. 1991. “VideoWhiteboard: video shadows to support remote collaboration”. In *Proc. ACM SIGCHI Conference on Human Factors in Computing Systems*, pp. 315 – 322, April 1991.

Vidot, N., Cart, M., Ferrie, J., and Suleiman, M. 2000. “Copies convergence in a distributed realtime collaborative environment”. In *Proc. ACM Conference on Computer-Supported Cooperative Work*, pp. 171 – 180, December 2000.

Wang, X. 1996. “Tabular abstraction, editing, and formatting”, *PhD thesis*, University of Waterloo, Ontario, Canada, 1996.

WARP 2006. Wide Area Resource Programme (WARP), St Andrews University, United Kingdoms. http://distsyst.dcs.st-and.ac.uk/warp/warp.html, last accessed: February 2006.

Xia, S., Sun, D., Sun, C. and Chen, D. 2005a. “A collaborative table editing technique based on transparent adaptation”. In *Proc. the 13th International*

Conference on Cooperative Information Systems (CoopIS 2005), LNCS, Springer Verlag, vol. 3760, pp. 576 – 592, November 2005.

Xia, S., Sun, D., Sun, C. and Chen, D. 2005b. "Supporting workspace-mediated interaction in collaborative presentations with CoPowerPoint". In *Proc. The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, December 2005.

Xia, S., Sun, D., Sun, C. and Chen, D. 2005c. "Object-associated telepointer for real-time collaborative document editing systems". In *Proc. The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, December 2005.

Xia, S., Sun, D., Sun, C. and Chen, D. 2005d. "Collaborative object grouping in graphics editing systems". In *Proc. The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, December 2005.

Xia, S., Sun, D., Sun, C. and Chen, D. 2006. "An integrated session and repository management approach for real-time collaborative editing systems". In *Proc. the Fourth International Conference on Creating, Connecting and Collaborating through Computing (C^5 2006),* January 2006.

Xia, S., Sun, D., Sun, C., Chen, D. and Shen, H. 2004a. "Leveraging single-user applications for multi-user collaboration: the CoWord approach". In *Proc. ACM 2004 Conference on Computer Supported Cooperative Work*, pp. 162 – 171, December 2004.

Xia, S., Sun, D., Sun, C., Chen, D. and Shi, Y. 2004b. "Interactive Presentation with CoPowerPoint". *The Sixth International Workshop on Collaborative Editing Systems*, pp. 437 – 446, 6-10 November 2004.

Zhang, K., Pande, S. 2005. "Efficient Application Migration under Compiler Guidance". In *Proc. 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pp. 11 – 20, June 2005.

Index