Leveraging Single-user Applications for Multi-user Collaboration: the CoWord Approach

Steven Xia, David Sun, Chengzheng Sun, David Chen
School of Computing and Information Technology
Griffith University
Brisbane, Qld 4111, Australia
{Q.Xia,D.Sun,C.Sun,D.Chen}@griffith.edu.au

Haifeng Shen
School of Computer Engineering
Nanyang Technological University
Singapore 639798
ASHFShen@ntu.edu.sg

ABSTRACT

Single-user interactive computer applications are pervasive in our daily lives and work. Leveraging single-user applications for multi-user collaboration has the potential to significantly increase the availability and improve the usability of collaborative applications. In this paper, we report an innovative transparent adaptation approach for this purpose.

The basic idea is to adapt the single-user application programming interface to the data and operational models of the underlying collaboration supporting technique, namely Operational Transformation. Distinctive features of this approach include: (1) Application transparency: it does not require access to the source code of the single-user application; (2) Unconstrained collaboration: it supports concurrent and free interaction and collaboration among multiple users; and (3) Reusable collaborative software components: collaborative software components developed with this approach can be reused in adapting a wide range of single-user applications. This approach has been applied to transparently convert MS Word into a real-time collaborative word processor, called CoWord, which supports multiple users to view and edit any objects in the same Word document at the same time over the Internet. The generality of this approach has been tested by re-applying it to convert MS PowerPoint into CoPowerPoint.

1. INTRODUCTION

Single-user interactive computer applications are pervasive in our daily lives and work. Plain text editors (e.g. vi, emacs), graphic drawing tools (e.g. PhotoShop), word processors (e.g. Word, WordPerfect), spreadsheets (e.g. Excel), slide authoring and presentation tools (e.g. PowerPoint), web design tools (e.g. FrontPage, Dreamweaver), and CAD/CASE systems (e.g. AutoCAD), are just some of the commonly used single-user interactive applications.

With the increasing importance of using computers to support collaborative work [10], it is natural to expect existing single-user computer applications to play an important role in supporting collaboration. The benefits of using single-user interactive applications for collaboration have long been recognized and explored. Representative early systems include Dialogo [15], XTV [1], SharedIX [8], MS NetMeeting,1 and SunForum,2 etc. These early systems have helped popularize collaborative use of single-user applications, but their capability in supporting collaboration is limited. For comprehensive reviews and critiques of these early systems, the reader is referred to [2, 24].

Research on collaborative systems in the past decades, on the other hand, has made significant progress in developing collaborative techniques to support both real-time and non-real-time collaboration [2, 3, 5, 7, 9, 12, 14, 17, 18, 20, 22, 26, 27, 29]. However, state-of-the-art collaboration techniques are only available in research prototypes. These prototypes serve mainly as research vehicles, and, as such, have limited functionalities in supporting conventional single-user interactive activities.

A significant challenge for collaborative computing research is to build computer applications that combine mature conventional single-user functionalities and interface features with state-of-the-art collaborative techniques, so that users can use the same computer applications to support their individual as well as group work. The goal of our research is to investigate and develop innovative approaches and techniques to address this challenge.

One possible approach is to further develop research prototypes to gain adequate conventional functionalities comparable with corresponding commercial single-user applications. One problem with this approach is that the resource requirement for re-inventing conventional functionalities in collaborative applications can be prohibitive since conventional functionalities are significantly richer than collabo-

---

rative functionalities in most applications. Another problem is that computer applications built in this way may not be compatible in functionality and interface with commonly used single-user counterparts. Consequently, users are unlikely to give up their favorite single-user applications to switch to new collaborative applications just for less frequently-used collaborative features [10].

Another approach is to modify existing open-source single-user applications to integrate collaborative techniques. This can be achieved with relatively small implementation efforts due to the relatively small size of collaborative functionalities compared with conventional single-user functionalities. However, this approach is not applicable to many widely-used commercial off-the-shelf single-user applications based on proprietary source code.

Software vendors of commercial single-user applications may revise their products in future versions to integrate collaborative techniques, but that does not address an immediate need for collaborative use. Indeed, many existing single-user applications may never be revised to support collaboration. Moreover, from a system design point of view, it is more advantageous to separate single-user applications from collaboration concerns, as shall be shown in this paper.

An innovative transparent adaptation approach to leveraging single-user applications for collaborative use is proposed in this paper. The basic idea is to adapt the single-user application programming interface to the data and operational models of the underlying collaboration supporting technique, namely Operational Transformation (OT) [28]. Distinctive features of this approach are as follows. First, this approach is application transparent in the sense that it does not require access to the source code of the single-user application, which may be a commercial off-the-shelf single-user system (without source code publicly available), an existing open-source single-user application (with source code publicly available), or a new single-user functional component in a collaborative application. Second, this approach supports unconstrained collaboration in the sense that it allows multiple users to interact with the shared application concurrently and freely. Third, collaborative software components developed with this approach are reusable in adapting a wide range of single-user applications.

We have chosen MS Word 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 MS Word provides a rich set of complex and interesting data types and operations for investigation. Our goal is to convert MS Word into a real-time collaborative word processor, called CoWord, that allows multiple users to view and edit any objects in the same Word document at the same time over the Internet. This paper reports some of the research findings from the CoWord project.

The rest of this paper is organized as follows. First, prior representative work in supporting collaborative use of single-user applications are reviewed, and the CoWord approach and design goals are discussed in Section 2. Then, main issues in applying OT to Word are discussed in Section 3. Next, transparent adaptation of the Word API to meet the requirements of OT is discussed in Section 4. Based on the transparent adaptation approach, the CoWord architecture and system components are discussed in Section 5. In Section 6, we discuss the generality and limitations of the CoWord approach, and give an outline of some ongoing and future work. Finally, major research findings and contributions reported in this paper are summarized in Section 7.

2. PRIOR WORK AND COWORD GOALS

2.1 Generic application sharing environments

NetMeeting and SunForum are commercial and representative generic application sharing environments, which allow existing single-user applications to be shared without any assumption about how the shared application was written, or requiring any application-specific treatment. These systems support "view-sharing" in a manner referred to as strict WYSIWIS (What You See Is What I See), where the users see exactly the same view of the shared application at the same time [24]. Apart from supporting a single-view, they also support a single-actor interaction paradigm; multiple users can view the same display of the shared application at the same time, but only one user (the actor) can interact with the shared application at any instant of time. To gain control over the shared application, a user must first obtain the floor, which may become a "sequential bottleneck" for some collaborative work. If the shared application is not running on the current actor's site, the actor's local response shall degrade, particularly in the Internet environment.

Strict WYSIWIS and sequential interaction with the shared application are useful and effective for tightly coupled collaborative work, but too restrictive for collaborative work with any degree of independency. Many researchers [24, 19, 14, 22, 2, 7, 9] have cited strict WYSIWIS, sequential bottleneck, and poor responsiveness as major shortcomings of some early application sharing systems, since they hinder multi-user free interaction and concurrent work, and hence decrease the productivity and quality of group work in some collaborative activities. Some of these shortcomings can be attributed to the use of the centralized architecture (responsible for poor responsiveness) and the use of simple application-level locking based on floor control (responsible for the sequential bottleneck) [2]. Strict WYSIWIS is a result of the desire to share views and the inability to selectively share different aspects of the views. Moreover, the goal to provide a generic environment for sharing any single-user application has prohibited the use of application-specific semantic information. Without application semantic information, optimistic consistency maintenance techniques like OT cannot be utilized and concurrent interaction with the application cannot be supported even if the replicated architecture were adopted.

2.2 Component-replacement approach

Flexible JAMM [2] is a component-replacement approach for transparent sharing of single-user applications. This approach is based on automatic replacement of the shared application's single-user components with multi-user versions at runtime. Unlike the generic application sharing environment which imposes no requirement on the application for sharing, this approach requires the application and its underlying execution environment to meet certain conditions, including capabilities for run-time component replacement, dynamic binding, and user input events interception and replay [2].

This approach has been applied to some Java applications based on the Swing and Java Object Serialization (JOS) library that meet all Flexible JAMM requirements. Appli-
cations based on Flexible JMM can adopt a distributed replicated architecture, intercept user input events before they are delivered to the application, extract the application’s semantic information from input events, convert input events into suitable high-level operations, and use collaborative techniques for achieving fast local response, consistency maintenance, relaxed WYSIWIS, and group awareness. In addition, Flexible JMM made contributions in explicitly addressing issues related to externalities (e.g., files and network connections) in replicated real-time collaborative applications [3]. The main limitation of this approach is the set of single-user applications and execution platforms that meet its requirements is fairly small [2] and does not include most commercial off-the-shelf single-user applications.

2.3 Transparent adaptation approach

The Transparent Adaptation (TA) approach, developed from the CoWord work, is based on the use of the shared application and its execution environment’s API (Application Programming Interface) to intercept the user’s local inputs, convert these inputs into abstract operations, manipulate these operations by collaborative techniques, and replay the modified operations to the application at remote collaborating sites. This approach does not require access to the single-user application’s source code (so it is transparent), but it requires the shared application’s API to be adaptable to the data and operational models of the underlying collaborative techniques.

Unlike the generic application sharing environment which does not require any application-specific treatment, the TA approach requires each application to be adapted before being shared. The benefit of this application-specific adaptation is that it allows the use of application semantic knowledge and optimistic concurrency control techniques (like OT) to consistently manipulate and interpret the user’s interactions in different contexts. Combined with the replicated architecture and suitable collaborative techniques, the TA approach is able to achieve high responsiveness, concurrent work, relaxed WYSIWIS, and group awareness.

The TA approach and the CoWord work have drawn inspiration from various sources. First, our work in consistency maintenance and group undo techniques [28, 29, 27, 26, 4, 23] and in building collaboration-aware systems has provided us with insights, experiences, and motivation to apply these techniques to existing single-user applications. Second, prior work on application sharing [1, 2, 3, 8, 15, 16] has demonstrated the usefulness of transparent sharing and provided valuable experiences and lessons for later work. Third, software developers’ practice in using the API to enhance the functionality of an existing application has given the hint to use the API for sharing. Last but not least, the ICT (Intelligent Collaboration Transparency) work [17] has provided insights in intercepting and replaying user input events for transparent sharing and in heterogeneity and inter-operability issues that arise from sharing different single-user applications in the same session.

2.4 CoWord design goals

Based on the TA approach, CoWord has been designed to meet the following goals:

1. Application compatibility: CoWord should retain the user interface, functionalities, and document formats of MS Word.

2. Unconstrained collaboration: CoWord should allow multiple users to edit any Word objects (e.g., characters, graphic lines, ClipArt pictures, etc.) at any time, and to undo any Word operations at any time.

Application compatibility ensures that CoWord inherits the conventional single-user functionalities and the “look-and-feel” from MS Word. Unconstrained collaboration means no specific collaboration style or structure is imposed on users, giving users complete freedom in their ways of using CoWord. This also implies relaxed WYSIWIS and concurrent work (the multi-actor interaction paradigm).

The key to achieving unconstrained collaboration is the use of OT as the underlying core collaborative technique. OT is a technique for supporting consistency maintenance and group undo in a wide range of editor-like applications [5, 6, 29, 28, 2, 26, 4, 17]. The major technical challenge here is how to apply OT to MS Word, which has far more complex data and operational models than plain text editors.

It should be pointed out that the focus of the current CoWord work and this paper is on issues related to achieving data-sharing of Word, i.e., the user’s interactions are replayed at remote sites for sharing only if they are related to data manipulation and thus related to consistency maintenance. The user’s other interface actions, such as moving the cursor, opening/closing/scrolling windows, viewing hover text, etc., are not shared by other users in the current CoWord system. However, the basic TA approach does allow selective sharing of various aspects of the single-user application. Work is on the way to build a variety of sharing capabilities (including view-sharing) and interaction paradigms/policies (including sequential interaction) on top of the CoWord unconstrained collaboration mechanisms.

3. ISSUES IN APPLYING OT TO WORD

3.1 OT basics

The OT component in a collaborative editor is a complex system, but the basic idea of OT can be illustrated by using a simple text editing scenario as follows. Given a text document with a string “abc” replicated at two collaborating sites; and two concurrent operations: $O_1 = \text{Insert}[0, \text{"x"}]$ (to insert character “x” at position “0”), and $O_2 = \text{Delete}[3, \text{"c"}]$ (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: $O_2' = \text{Delete}[4, \text{"c"}]$, whose positional parameter is incremented by one due to the insertion of one character “x” by $O_1$. Executing $O_2'$ on “labc” shall delete the correct character “c” and the document becomes “labd”. However, if $O_2$ is executed without transformation, then it shall 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 [28].

Due to the simplicity and intuitiveness of text documents and operations, collaborative text editing scenarios have often been used in OT research literature to illustrate some of the very intricate OT technical problems and algorithms [6,
The common use of text-specific examples has played an excellent role in the development and explanation of OT techniques, but also unfortunately incurred some OT-misconceptions, which obscured the value and power of OT and hindered the application and advancement of OT.

For example, the positions of the two concurrent editing operations in the previous scenario are separated by only a small number (three) of characters “abc”. Misinterpretation of this kind of simple scenario had incurred the following misconception: OT is needed and useful only when users are concurrently editing in adjacent areas, e.g. the same word, the same line, or the same paragraph, etc. In fact, the applicability of OT is not related to the number of characters/objects separating concurrent operations. As shall be shown in this paper, OT is needed and useful even if concurrent editing activities are separated by millions of objects in large Word documents.

To identify and resolve the technical issues involved in applying OT to Word, it is important to rectify OT-misconceptions with OT-essentials that are independent of text editing, and to have a good understanding of Word documents and operations from the OT’s point of view.

3.2 Understand Word from OT’s point of view

3.2.1 Word document and OT addressing scheme

All characters in a plain text document are presented at the user interface in a sequence. In other words, the user’s view of a plain text document is a sequence of characters. This user’s view of the text document happens to match the OT’s view of the document since the two OT-supported operations Insert and Delete are defined on a sequence of characters. This coincident matching of the user’s view and the OT’s view of the text document has led to the following misconception: OT is only applicable to documents consisting of objects presented at the user interface sequentially, e.g. in a sequence of characters, sections, chapters, or frames, etc.

A Word Document, when viewed by a user, does not always psychologically map well into a linear sequence of objects. For example, graphical objects may appear at any position in the document’s two-dimensional display space. Furthermore, they can be moved freely from one location to another without the notion of any sequential ordering. As shown in Fig.1, the user’s view of a Word document consists of some sequences of formatted character objects, e.g. “CoWord, a collaborative word processor”; graphical objects that are inline with a sequence of characters, e.g. the “Welcome” ClipArt object is inline with the sequence of characters “To CoWord”; and graphic objects that are floating in the two-dimensional space, and may overlap with each other, e.g. the textbox with text “Word” is on top of another ClipArt object in Fig.1.

This irregular and arbitrary presentation/view of data objects in a Word document appears to be a major obstacle (due to the above misconception) for applying OT to Word documents, but in-depth investigation in the CoWord project has discovered the following: all data objects of a Word document can actually be accessed by their positional references in a linear address space from an adapted Word API (discussed in detail in the next section), which meets the data modelling requirement of OT.

As shown in Fig.1, objects’ references in the linear address space do not necessarily follow the same order as they appear at the user interface. For example, the two objects referred to by positions 12 and 13 in the linear address space are actually located at the user interface after (on the right of) another two objects referred to by positions 14 and 15. When the user draws a new graphic object (the cross “+” sign) in the document (the user’s view in Fig.1-(b)), this object is assigned position “14” in the linear address space.
Meanwhile, other objects with positional references on 14 or higher are shifted to the right by one position, as shown in the adapted API's view in Fig.1-(b). If an object is removed from the document, its position in the linear address space shall be removed and all other objects with positional references on the right of the removed object shall be shifted to the left by one position (not shown in Fig.1).

The above analysis of the Word document from the OT's point of view is an important discovery from this research because it not only established a bridge over the conceptual gap between the user's view and the OT's view of the complex Word document, but also advanced our understanding of an essential requirement for OT: OT requires the objects in a document to be addressable by positional references in a one- or multi-dimensional [4]address space, but does not require the objects in a document to be presented, or viewed by the user, as a sequence at the user interface.

3.2.2 Word operations and OT operations

In plain text documents, characters have no updatable attribute. Therefore, two primitive editing operations Insert and Delete are sufficient to support plain text editing, and all other high-level compound editing operations can be mapped into these two primitive operations. Existing OT systems for plain text editors are defined on Insert and Delete only.

In Word documents, however, objects may have various attributes (e.g. size, color, style, boldness, etc.). A user can not only insert/draw new objects and delete objects, but also update the attributes of existing objects. For example, the font size of a character can be changed from 10pt to 12pt, the color of a graphic circle object can be changed from red to blue, and a graphic square object can be moved from one position to another in the document, etc. A large number of Word editing operations have the effect of updating some attributes of existing objects. To provide adequate support for collaborative word processing, it is necessary to extend the set of OT-supported primitive operations to include a new and generic operation Update. The issues and solutions involved in extending OT for Updates are discussed in another paper [30]. In this paper, we focus on the issues involved in adapting high-level user-oriented word processing operations to OT-supported primitive operations Insert, Delete, and Update.

One may wonder why not extend the basic OT transformation functions to directly transform every pair of application operations? Suppose an application supports n different operations, then n × n different transformation functions can be defined to support this application, as suggested by some early work on OT [6]. A major problem with this application-oriented operational transformation strategy is that application level transformation functions are too difficult to design and too complex to ensure correctness.\(^3\) Another problem with this approach is that the designed transformation functions are application-specific and not reusable in other applications with slightly different application semantics. Therefore, we take the strategy of keeping the OT core small and application independent (to reduce its complexity and increase its reusability) by designing transformation functions on three primitive operations Insert, Delete, and Update only, and translating application level operations into these primitive operations for transformation. From our experience with Word, all Word operations related to document data manipulation can be mapped into these three primitive operations.

4. ADAPTATION OF WORD API

MS Word provides a comprehensive API which conforms with Component Object Model (COM) Automation [15]. With this Word API, developers can change the behavior of Word, enhance Word's functionality, or incorporate Word into other applications. In particular, this Word API provides a high level interface for accessing and manipulating data objects in a Word document. One important element of our transparent adaptation approach is the intermediate Adaptation Layer (AL) between the Word API layer and the OT layer, as shown in Fig.2.

At the Word API Layer (Fig.2-(a)), a Word document is modelled by a root Document object, and a contiguous area of a document is modelled as a Range object. From the Range object, all user-generated data objects can be accessed. Data objects of various types (e.g. Text, ClipArt pictures, Drawing objects, Equation Editor objects, and WordArt objects, etc.) are modelled by some basic objects, including Text\(^4\) (e.g. a sequence of formatted characters), InlineShape (e.g. a ClipArt picture embedded in a sequence of characters), or Shape (e.g. a drawing graphic object), etc. Different types of object can be manipulated in different ways, so numerous Word API functions have been provided to achieve their corresponding data-type-dependent editing effects.

The Word API also provides multiple addressing schemes for accessing data objects in the document. For example, all text objects and inlineShape objects in a contiguous range of a document can be accessed by their positional references in a global linear sequence (denoted by an array of objects in Fig.2-(a)). Floating objects can be accessed in multiple ways, e.g. by unique names (e.g. S1, S2, S3, S4, S5, S6 in Fig.2-(a)), by Z-orders (indicated as dashed lines of arrow in Fig. 2-(a)), or by anchors in the global linear sequence (indicated as solid lines of arrow in Fig. 2-(a)). Creating or removing a floating object results in inserting or deleting its anchor in the linear sequence.

At the bottom OT Layer (Fig.2-(c)), there are only three Primitive Operations (PO): Insert, Delete, and Update, which are defined on a single linear addressing scheme (indicated as an array of positional references in Fig.2-(c)). Moreover, POs are independent of data types: the objSeq parameter in Insert and Delete is generic for all types of data object, and the key parameter in Update is generic for all types of attribute.

The task of the Adaptation Layer (Fig.2-(b)) is to bridge the data and operational gap between the Word API and OT by means of a collection of Adapted Operations (AO). In contrast to the Word API which provides multiple ways of accessing data objects in a document, AOs are defined on a single linear address space (using a pos parameter and indicated by an array of positional references in Fig.2-(b)). This AO address space can be directly mapped into the global lin-

\(^3\)To get an idea about the complexity of designing just two string-wise editing operations Insert and Delete, the reader is referred to [29].

\(^4\)In fact, text is treated as part of the Range object, rather than as a separate object. We treat text as an object for the sake of convenience.
Figure 2: Three layers in the transparent adaptation approach.

ear sequence provided by the Word API Range object. In some sense, the Adaptation Layer acts like a filter that selectively passes a linear addressing scheme from the Word API Layer to the OT Layer.

Moreover, AOs extract a subset of functionalities from the Word API which are relevant to manipulating document state and hence to maintaining maintenance. Similar to the Word API, AOs are data-type-dependent, as indicated by an array of object type symbols associated with their positional references in Fig. 2-(b). AOs are named and grouped according to the data 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 and Frame objects in the Word API), and the floatingObj operation group (corresponding to the Shape object in the Word API), etc. This dimension of operation naming and grouping helps the operation mapping between AO and PO.

AOs can be classified into two categories: (1) basic AOs, such as Insert-floatingObj, Delete-floatingObj, and Resize-floatingObj (an Update for inline objects), etc.; and for the floating object operation group, we have Insert-floatingObj, Delete-floatingObj, and Move-floatingObj (an Update for floating objects), etc. This dimension of operation naming and grouping helps the operation mapping between AO and PO.

AOs can be classified into two categories: (1) basic AOs, such as Insert-floatingObj, Delete-floatingObj, and Resize-floatingObj (an Update for inline objects), etc.; and for the floating object operation group, we have Insert-floatingObj, Delete-floatingObj, and Move-floatingObj (an Update for floating objects), etc. This dimension of operation naming and grouping helps the operation mapping between AO and PO.

In summary, the Adaptation Layer provides an intermediate level of abstraction to separate the single-user functional component from the generic multi-user collaboration component in the system, and to support the data and operational mapping between the single-user application (MS Word) API and the underlying collaboration technique (OT).
5. SYSTEM ARCHITECTURE

The CoWord system allows multiple users to edit the same Word document at the same time over the Internet, as shown in Fig. 3. A collaborative CoWord editing session consists of multiple CoWord instances. Each user interacts directly with a local CoWord instance, which maintains and manipulates a local copy of the shared Word document. Shared Word documents can be located in any collaborating user’s local file system and any collaborator with permission (e.g., password) to access the shared documents can start or join a collaborative editing session. In the Internet-based CoWord Demo, only one Collaborative Document Repository Manager (CDRM) is used to provide world-wide users with remote access to the Word documents stored on a single machine hosted by Griffith University, Brisbane, Australia. This CDRM can be installed on any user’s machine to convert his/her local document repository (file system) into a shared document repository to support collaborative editing. The CDRM plays an important role in managing shared documents (a part of the externalities discussed in [3]) and collaborative sessions across the Internet, but a detailed discussion of CDRM is beyond the scope of this paper.

Based on the transparent adaptation approach discussed in Section 4 and Fig. 2, a CoWord instance is composed of three components (see Fig 3):

Single-user Application (SA) (i.e. MS Word) provides the conventional single-user functionalities and interface features. This component is completely collaboration-unaware (or collaboration-transparent).

Generic Collaboration Engine (GCE) provides application-independent collaboration capabilities. This component is fully collaboration-aware, but completely unaware of the single-user application.

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

The CA component consists of the following major modules:

API-AO Adaptation This module is responsible for translation between Word and Windows APIs and AOs (Adapted Operations) defined in Section 4 and illustrated in Fig. 2. In addition, this component also provides an adapted interface for other CA modules to access Word and Windows APIs. This module hides application-specific APIs from the rest of the system.

AO-PO Adaptation This module is responsible for translation between AOs used in CA and PO (Primitive Operations) used by OT (in GCE), as discussed in Section 4 and illustrated in Fig. 2. It also provides a common interface for other CA modules to access GCE.

Local Operation Handler (LOH) This module is responsible for intercepting local events and translating local events (e.g., key-down, key-up, etc.) into suitable AOs (e.g., Insert-text(), etc.). In the current version of CoWord, local events that are not related to document data modification (e.g., some cursor moves or window close events) are ignored by LOH. LOH makes use of application semantic information to select and translate input events into AOs. This module utilizes the API-AO Adaptation module to do its work. For example, LOH makes Word API calls to detect which object the user is accessing, to understand what operation the user is performing on the object, and to derive the parameters of this operation, including the positional references of the object in the linear addressing space, the inserted/deleted object or the updated object attribute (both new and old values). LOH also controls the granularity of AOs. For example, a sequence of character insertion events may be packed into a single string-wise insertion AO. Moreover, LOH makes use of the AO-PO module to translate the converted AO into suitable POs for consistency maintenance and group undo by GCE.

Remote Operation Handler (ROH) This module is responsible for receiving and processing remote operations (in the form of AO). It uses the AO-PO module to translate AOs into suitable POs for consistency maintenance or group undo by GCE and then calls the API-AO Adaptation module to interpret the transformed remote operations. It should be noted that ROH and LOH are implemented as two concurrent but mutually exclusive threads in CA: only one of them could be active at any instant of time to ensure atomicity of local and remote operations. LOH is given a higher priority when both threads are competing for the control of CA. When ROH is processing a remote AO, the local user interaction with the application is temporarily blocked or ignored by LOH. Moreover, ROH provides services to propagate local operations to remote sites.

Other modules include the Application Data Processing (ADP) module, which is responsible for processing application-specific data types. It makes use of API-AO Adaptation module to manipulate various types of data object transparently. It also provides a common interface to other CA and GCE modules for accessing and processing various application data types. The Collaboration Policy (CP) module is responsible for implementing application-specific collaboration policies, including various group modes and scopes [26], etc.

The GCE component contains a number of modules implementing a package of generic collaboration techniques. OT is at the core of supporting Consistency Maintenance (CM) and Group Undo (GU). Discussions on techniques related to OT-based CM and GU modules can be found in [26, 25, 23, 4, 30]. Another module in GCE is Session Manager (SM) which provides the client-side generic support for session management and allows collaborating users to join or leave a session at any time. The Group Awareness (GA) module and other modules are still under investigation and development using CoWord as the research vehicle.

To illustrate how CoWord components work together in processing an editing operation, consider the following simple scenario. Suppose a user uses the keyboard and/or mouse to create a graphic object in the local Word document, the following events shall occur at the local site: (1) The sequence of local input events (from the key-
board and/or mouse) are intercepted, performed immediately on the local document copy, and translated into an AO: Insert-floatingObj(pos, len, floatingObj) by LOH. (2) LOH calls AO-PO Adaptation to translate this AO into a PO Insert(pos, len, objSeq). (3) This PO is processed (e.g. timestamped) by OT and saved in OT’s history buffer. (4) The AO is attached with the same timestamp as its corresponding PO and propagated to remote sites.

When the AO Insert-floatingObj(pos, len, floatingObj) arrives at a remote site, the following shall happen: (1) The AO is received by ROH. (2) ROH calls AO-PO Adaptation to translate the AO into a PO Insert(pos, len, objSeq). (3) This PO is transformed by OT for consistency maintenance and saved in OT’s history buffer. (4) The transformed PO is used to transform the original AO into a new AO by AO-PO Adaptation. (5) ROH calls API-AO Adaptation to interpret the new AO by means of a sequence of Word API calls to the remote Word document.

6. DISCUSSIONS

6.1 Generality of the CoWord approach

6.1.1 Reusable system architecture and collaboration components

The transparent adaptation approach has led to a novel collaborative system architecture consisting of three components: the Single-user Application (SA) component, the Collaboration Adaptor (CA), and the Generic Collaboration Engine (GCE). The use of CA hides application-specific issues from GCE, facilitates independent debugging and testing of GCE, and increases 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[28, 29, 26, 30]. To apply GCE to a new SA, one only needs to design and implement a new CA for the target SA.

As a follow-up of CoWord, we have applied the same approach to convert MS PowerPoint into CoPowerPoint. The successful construction of CoPowerPoint is a testimony to the generality of the transparent adaptation approach and the reusability of the system architecture and the GCE component. From CoPowerPoint, we learnt that PowerPoint data objects can be mapped into a tree of multiple linear addressing domains (when viewed from the PowerPoint COM API). The support for this tree-based data modeling could be built in the CoPowerPoint Adaptor without changing the GCE component developed from the CoWord project. However, since the tree of multiple linear addressing domains is more general than the single linear addressing space, it is beneficial to incorporate this general model into the GCE component.

component. With this extension, a wider range of applications can be directly supported by GCE without the need to build specific tree-supporting mechanisms in their CAs. Therefore, we made this extension to GCE and the extended GCE is now being shared by both CoWord and CoPowerPoint. Work is on the way to apply the CoWord approach, architecture, and the GCE component to a selected set of single-user applications in different application domains and different platforms.

6.1.2 Unification of collaboration-awareness and collaboration-transparency

Collaborative systems have been traditionally classified into two different categories [16, 2]; collaboration-transparent systems that provide the shared use of single-user applications through mechanisms that are unknown to the applications and their developers; and collaboration-aware systems that are specifically designed for supporting collaborative work, and collaboration mechanisms were known to both the applications and developers. The CoWord work can be seen as a unified approach to designing both collaboration-transparent and -aware systems: the SA component in the CoWord system architecture can be a commercial off-the-shelf single-user application, or a newly designed single-user functional component in a collaboration-aware system. This new single-user functional component can be designed and implemented in the same ways as a stand-alone single-user application without any concerns about collaboration, except that it provides an API suitable for collaboration adaptation.

The clear separation of single-user functionality and multi-user functionality helps to reduce the complexity of collaborative system design, and increase modularity and reusability of system components. The performance cost of this design approach is acceptable. From our experience in building and using CoWord, the performance of the multi-user application (CoWord) is comparable to that of the existing single-user application (MS Word) because the time taken by collaboration components GCE and CA is fairly small compared to the time taken by the complex application itself. Under the transparent adaptation approach and system architecture, the traditional distinction between collaboration-transparent and collaboration-aware applications has now 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.

6.2 Requirements, limitations, and future work

The CoWord approach requires the single-user application and its execution environment to provide a suitable API (1) which can be used to intercept and replay user input events, and (2) whose data and operational models are adaptable to that of the underlying OT technique. The first requirement is generally satisfactory 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 modern office software suites (e.g. MS Office\(^7\) and OpenOffice\(^8\)). In general, we expect these requirements to be satisfactory by a wide range of single-user editor-like applications, including various word processors, graphic drawing and design tools, and CAD/CASE systems.

Existing single-user APIs, however, were not designed with regard to supporting transparent sharing, so the collaboration adaptation process may not be always straightforward and sometimes suboptimal strategies may have to be used. One important next step of our research is to identify and define a set of essential and versatile CA(Collaboration Adaptation)-oriented API functionalities, which can be used to inform single-user API designers to provide better API support for collaboration adaptation. When CA-oriented functionalities become a standard part of single-user API packages, more single-user applications shall become readily convertible for collaborative use.

The underlying collaborative technique in the CoWord approach is OT. We are unable to define a hard boundary for OT’s application scope since we learnt from the past experience that the power of OT was often underestimated and this boundary has been continuously expanding as the exploration of OT marches into new territories.

There are other important issues for supporting real-time collaboration, which have not been adequately addressed by our work so far. One of them is workspace awareness – the up-to-the-moment understanding of other persons’ interaction with a shared workspace [11]. The current CoWord system is limited to making the user aware of who else is in the current editing session, but provides no information about the view-ports and cursor positions of other collaborating users. Better workspace awareness support is needed to coordinate users’ activities in the unconstrained collaboration environment where users may freely view and work in any areas of the shared workspace. There have been some well-known techniques in this area, such as radar views [11], telepointers [21], multiuser scroll bars [22], etc. Using CoWord and CoPowerPoint as research vehicles, we are investigating how to incorporate known techniques and devise new techniques for supporting workspace awareness in transparently adapted applications.

Other ongoing and future work include: incorporation of flexible notification [23, 20] for supporting both real-time and non-real-time collaboration, integration of flexible fine-grain locking [25, 18] for supporting both syntactic and semantic consistency maintenance [28], investigation of externalities [3] and heterogeneity [17] for sharing single-user applications of different versions, vendors, or platforms, and usability study on collaborative applications based on the CoWord approach.

7. CONCLUSIONS

This paper contributes an innovative transparent adaptation approach to leveraging existing and new single-user applications for collaborative use. This approach has been applied to convert MS Word into CoWord – a real-time collaborative word processor, without changing the source code of Word. CoWord is able to retain the conventional single-user functionalities and interface features of Word, while supporting multiple geographically dispersed users to edit any objects in the same Word document at any time over the Internet.

The key to integrating unconstrained collaboration capabilities with the single-user application is an adaptation technique that bridges the data and operational modelling gap between the single-user application programming interface and the underlying Operational Transformation (OT) technique. Based on this approach, a collaborative system


architecture consisting of three components has been proposed: the single-user application, the application-specific collaboration adaptor, and the generic collaboration engine (with OT at its core). The separation of single-user functionalities and collaboration capabilities in this architecture serves to increase system modularity, reduce design complexity, and promote component reusability. The generality of this approach, architecture, and the generic collaboration engine component has been tested by re-applying them to convert MS PowerPoint into CoPowerPoint.

OT was known for its application in collaborative editing of plain text and SGML/XML documents. Prior to the CoWord project, it was unknown whether OT is applicable to collaborative editing of complex documents like Word. The CoWord work has demonstrated that OT is applicable not only to Word, but also to a range of commercial off-the-shelf single-user applications. The CoWord work and the exploration of OT are still ongoing.

ACKNOWLEDGEMENT

The authors are grateful to John Patterson and anonymous referees for their valuable comments and suggestions which have helped clarify some technical issues and improve the presentation of the paper.

8. REFERENCES