API Design Recommendations for Facilitating Conversion of Single-user Applications into Collaborative Applications

API Design Recommendations for Facilitating Conversion of Single-user Applications into Collaborative Applications

Kai Lin, David Chen, Geoff Dromey
School of Information and Communication Technology
Griffith University
Brisbane, QLD 4111, Australia
{K.Lin, D.Chen, G.Dromey}@griffith.edu.au

Steven Xia, Chengzheng Sun
School of Computer Engineering
Nanyang Technological University
Singapore, 639798
{StevenXia, CZSun}@ntu.edu.sg

***Abstract*—Recent advancements in collaboration technology have shown that it is possible to convert existing single-user applications into real-time collaborative applications without modifying the source codes of the single-user applications. Such conversion relies on the API (Application Programming Interface) provided by the single-user applications. Poorly designed APIs make such conversion difficult, inefficient, or even impossible. Until now, it is not well understood what features APIs should provide to facilitate conversion of single-user applications into collaborative applications. This paper presents recommendations of the features single-user application APIs should provide to facilitate conversion. The results are based on what we have learnt from converting Microsoft Visio, into real-time collaborative Visio (CoVisio), and our previous experience in building CoWord and CoPowerPoint.**

Keywords-API recommendations; collaborative system design; operational transformation

I. INTRODUCTION

With the increasing importance of using computers to support collaborative work, it is natural to expect existing single-user computer applications to play an important role in supporting collaboration. Leveraging single-user commercial systems for real-time multi-user collaboration has been a popular research topic for many years [3], [13], [15], [30]. So far, pioneer researchers have successfully enriched Microsoft Word and PowerPoint with collaborative functions, without modifying the source codes of MS Word and PowerPoint [30], [33]. This is achieved through the use of MS Word/PowerPoint's API (Application Programming Interface), to combine Word/PowerPoint with collaboration features.

APIs allow software developers to access and manipulate objects in a commercial application without knowing the implementation details of the application, which makes it possible to extend single-user commercial systems for multi-user collaborations. However, these APIs were designed without the knowledge of supporting collaboration. They may be unable to intercept and replay operations correctly in collaborative environments. Moreover, they may define data and operation models that cannot satisfy the requirements of a collaborative system. Even if methods could be investigated to bridge these gaps, APIs determine the complexity of these methods and required developing effort.

The needs and benefits to use existing single-user applications for collaboration have long been recognized [12], [15], [33], so that more and more application developers are considering how to support collaboration by providing suitable API functions. This benefits not only the collaborative system developers, but also the software vendors of commercial single-user systems, as these APIs may extend existing single-user commercial systems to multi-user collaborative versions. However, as single-user API designers are often not experts of collaborative systems, they may be uncertain about the requirements of multi-user concurrent systems. Hence, it is important for collaborative system researchers to provide recommendations for API design.

This paper presents recommendations for single-user application's API design based on the lessons learnt from developing a collaborative Visio system, called CoVisio, which enables a group of users to view and edit the same Microsoft Visio documents at the same time from different sites. Microsoft Visio is one of the most prevalent commercial single-user graphic editing systems, which can be used to create a wide variety of business and technical drawings. It is desirable to furnish single-user Visio system with multi-user collaborative functions, so that users can work collaboratively in groups to improve productivity.

CoVisio uses the API provided by Microsoft Visio to intercept and replay users' operations, so it requires no access or change to Visio's source codes. Visio API provides a rich set of graphic manipulation functions for software developers to manipulate Visio graphic objects in a very fine granularity. Visio API functions are very comprehensive and powerful, which makes Visio a suitable vehicle for studying API requirements.

The recommendations presented in this paper are helpful to both API designers and collaborative system designers. API designers can apply these recommendations in API designs to support collaborations. On the other hand, based on these recommendations, a collaborative system designer knows what key issues must be of concern when he/she investigates the API

provided by a commercial system to extend the system with collaborative functions.

The rest of this article is organized as follows. The next section introduces existing approaches to leverage single-user applications for multi-user collaboration, transparent adaptation strategy and the role of single-user application's API. The third section briefly introduces CoVisio. In the fourth section, we present API design recommendations for single-user applications, including the requirements for API functions to intercept and replay operations in concurrent environments and how to efficiently support data models adaptation. The last section is the conclusion of this paper.

II. BACKGROUND

Collaborative systems are groupware applications to support people working together in groups, such as electronic conferencing/meeting, collaborative CAD and CASE [28], [29]. The needs and benefits to use existing single-user applications for collaboration have long been recognized. Some approaches have been proposed to leverage single-user applications for multi-user collaboration. This section introduces these approaches, especially, transparent adaptation strategy and the role of single-user application's API.

A. Leveraging Single-user Applications for Multi-user Collaboration

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, such as Microsoft NetMeeting, and HP Shared X [10]. In these systems, the shared application is maintained at a single location, known as server site or central site. Other collaborating sites are known as client sites. User input events are forwarded to the server site. Then the server process running at server site will manipulate the shared data/documents according to the users' inputs. After that, the display graphical output is sent from the central shared application to client sites. The technique used to transmit and display graphical output from the central shared application is called *display broadcasting* [1].

These systems have many disadvantages. For example, only one participant can act at one time, which constrains the system concurrency. Moreover, they typically require higher network bandwidth to distribute graphical display information, and they impose strict What You See Is What I See (WYSIWIS), where the participants see exactly the same view of the shared application at the same time, which disallows independent work.

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. Shared documents are replicated at the local storage of each collaborating site, so that operations can be performed at local sites immediately and then propagated to remote sites. In contrast to the *display broadcasting* technique used in centralized systems, these systems adopt the *event broadcasting* technique, where the semantics of any local operation is marshaled into event-messages propagated to remote sites, so that the operation effects can be replayed and shown at remote sites.

To support *event broadcasting*, the semantics of any local operation must be correctly interpreted. If developers build a collaborative system from scratch, they define the semantics of any user operations. This will not be a big problem. However, for many widely used commercial off-the-shelf single-user applications whose source codes are not publicly available, correctly interpreting the semantics of user operations is a challenge.

One approach has been proposed which derives user-operations by *diffing* between document states instead of translating them from window events [15]. User-operations result in the state-change of a shared document. This approach propagates document-state differences caused by the execution of operations instead of the operations themselves between different collaborative sites. Document-state-change effects of concurrent operations are merged then applied to the document copies replicated at remote sites. This approach has two advantages, first of all, since it relies on state differences instead of specific user interfaces and operations that cause the differences, heterogeneous applications are allowed and there is no need for the infrastructure to understand every user-operation.

Moreover, this approach can be applied to text editors without providing APIs by simulating select/copy/paste events on the editors and accessing the clipboard. However, this approach mainly targets at supporting plain text editing applications. For a graphic editing application, where each graphic object has a rich set of attributes, detecting and expressing document-state differences could be complicated. Moreover, different graphic applications may define different document formats and different graphic object attributes, which makes the application of this approach more difficult. Even if this approach may have the potential to leverage various single-user applications for collaboration, till now its application is only in plain text editors.

Another approach adapts the existing single-user API to meet the data and operational modeling requirements of the underlying collaboration supporting technique [30], [33]. It uses APIs provided by single-user applications to intercept and replay user-operations. Compared with the *diffing* approach, this approach interprets the semantics of an operation according to its direct effects, such as moving/creating/deleting an object, etc., rather than the document-state differences before and after the execution of the operation. It relies on the APIs provided by the single-user applications, but it is a generic solution that will not be limited by the application domain of a system. Therefore, CoWord, CoPowerPoint and CoVisio are built based on this approach.

B. The Transparent Adaptation Approach

The method, adopted to leverage commercial single-user MS Word/PowerPoint and Visio for multi-user real-time collaboration, is known as *Transparent Adaptation* (TA) approach, which is based on the use of the single-user applications' APIs to intercept and replay users' operations, so

it requires no access or change to the applications' source codes (thus being transparent) [30], [33].

A TA based collaborative application is composed of three components. The first component is a *Single-user Application* (SA), i.e., MS Word/PowerPoint or Visio, which provides the conventional single-user functionalities and interface features. This component is completely collaboration-unaware. Another component is *Generic Collaboration Engine* (GCE), which provides application-independent collaboration capabilities. This component is fully collaboration-aware, but completely unaware of SA. GCE is generic, but SA is not. SAs may define different data and operation models. Therefore, the third component, *Collaboration Adapter* (CA), is implemented to adapt application-specific SA to generic GCE. CA provides application dependent collaboration capabilities and is aware of both the single-user and multi-user collaboration applications.

The interactions between the three components in processing an editing operation can be illustrated based on the following simple scenario in a CoVisio application, as shown in figure 1.

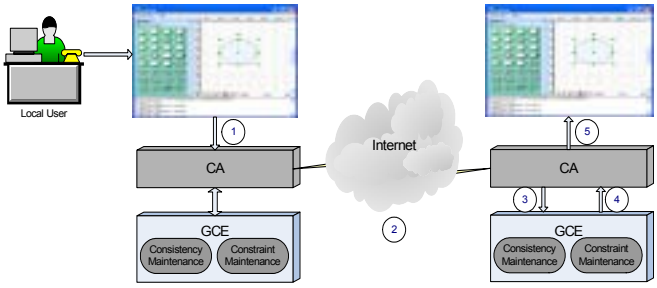

Figure 1. The interactions between CoVisio components

Suppose a user uses the keyboard and/or mouse to create a graphic object in a shared Visio document, the following events shall occur at the local site:

(1) Once the operation is performed on the local document, the operation semantics is sent to CA by SA.

(2) CA marshals the operation information into messages sent to remote sites.

When the messages arrive at a remote site, the following shall happen:

(3) CA un-marshals the messages and passes the operation information to GCE.

(4) The operation is processed by GCE for consistency and constraint maintenance. After that, the processed operation is sent back to CA.

(5) Suitable SA API functions are invoked by CA to apply the operation to the shared document replicated at the site.

C. The Role of Single-user Application's API

APIs play an important role in TA-based collaborative applications, as they provide an approach to access and manipulate objects in single-user applications without knowing the implementation details of these applications. Two crucial functions provided by APIs in collaborative applications are intercepting and replaying operations. For example, once a user updates the color of object *A* to *red* at a site in a CoVisio application, the operation semantics will be intercepted by Visio API, and marshaled into a message sent to remote sites. Once the message arrives at a remote site, it is un-marshaled and a Visio API function will be invoked to color the replica of *A* at the site to *red*.

On the other hand, not all the APIs provided by single-user applications can efficiently support collaboration. For instance, the API provided by a commercial system may define quite different operation and data models from the ones required by the GCE component of a collaborative system, which makes CA component difficult to bridge these gaps. In the worst case, a badly designed API cannot be applied to intercept and replay operations correctly in concurrent environments. It is the API provided by a single-user commercial system that determines the complexity and workload to transparently leverage the single-user system for multi-user collaboration.

III. CoVisio

CoVisio is built by extending single-user Microsoft Visio into a multi-user collaborative application, so that a group of users can use MS Visio to view and edit the same Visio documents at the same time from different sites. It is implemented in the programming language C# based on TA approach without knowing or modifying Visio source code. The interface of CoVisio is shown in figure 2.

Figure 2. The CoVisio interface

Compared with CoWord and CoPowerPoint, CoVisio is new in two aspects. First of all, a type of constraints, *formulas*, is defined in Visio to express the attributes of a graphic object and the relations between different graphic objects. The ability to describe shapes with *formulas* opens many possibilities for making shapes behave in complex and sophisticated ways. Accordingly, both constraints and consistency are maintained in CoVisio (Constraint maintenance in concurrent environments is beyond the scope of this paper). Moreover, the CA of CoVisio is quite different from the ones used in CoWord and CoPowerPoint, as it is based on Visio API. Visio API

provides a rich set of graphic manipulation functions for software developers to manipulate Visio graphic objects in a very fine granularity. Visio API functions are comprehensive and powerful, which makes Visio a suitable vehicle for studying API requirements.

IV. API Design Recommendations For Single-user Applications

APIs enable one application to access and manipulate the objects in another application without knowing the implementation details of the latter. They play a very important role in leveraging single-user commercial applications for multi-user collaborations, as all the interactions between a single-user commercial application and other components of a collaborative system are via API.

From our experience of converting Visio to CoVisio, we have come to know what aspects of the API provided by Visio make it simple or difficult to convert to CoVisio. We summarize our findings into recommendations for API design in the following subsections.

A. API's Role in Operation Interception

The two fundamental functions of APIs in collaborative systems are to intercept local operations and to replay remote operations. Without knowing the semantics of a user-operation, a local site cannot send suitable messages to remote sites to replay the operation there.

There are two types of API can be used to intercept an operation applied to a specific application, the API provided by the operating system and the API provided by the application.

To intercept an operation based on the API provided by the operating system, a program is implemented to intercept any user keyboard or mouse operation targeting a specific single-user application. Once such an operation is intercepted, its semantics is explained according to the keyboard inputs, the position of the cursor or the states of mouse buttons. Then the operation is passed to the local application for execution. Furthermore, its semantics is propagated to remote sites, so that the operation can be replayed there.

This approach has two advantages. It is a generic method that can intercept any operation applied to any application. Moreover, it can intercept operations before they take effects, which gives a collaborative system a chance to perform some operations, such as saving the values of some internal variables, before the execution of a user mouse/keyboard operation. However, this method has a drawback in that it involves a complex interpretation process to get the accurate operation semantics. For example, if a user resizes a shape by a mouse operation in Visio, he/she may change a lot of graphic attributes of the shape, such as width, height, geometrical center and the positions of some vertexes of the shape. It is hard for the APIs provided by an operating system, rather than the API provided Visio itself, to tell all these effects. Thus, this operation-interception method is not suitable for systems that require fine-granularity object-manipulation, such as graphic editing systems.

Another method is to interpret the semantics of an operation applied to an application by the application itself, which is adopted by CoVisio. User mouse/keyboard operations are directly inputted to single-user Visio application. To interpret the effects of user-operations, Collaboration Adapter (CA) component of CoVisio registers some event handlers in the single-user Visio application via Visio API. Therefore, when the events CoVisio interested arise, the single-user Visio application would automatically inform CA. For example, CA registers *ShapeAdded* event handler on each *page* object, so that each time a shape is added into a drawing page, CA will receive the detailed information of where and what a shape is created. Using this method, operations can be interpreted according to their effects. For example, no matter users change the *width* of an object to *w* via Visio GUI by mouse/keyboard operations, or through Visio *ShapeSheet* window by editing the formula that constrains the object's *width* attribute, the same event will be sent from Visio to CA.

Moreover, using this method, we can filter unimportant operations and only concentrate on the operations or operation effects we are interested (i.e. events triggered by unconcerned operations will not be handled). For example, a local *mouse-move* operation will not be intercepted. However, the local operations that update graphic attributes of graphic objects must be replayed at remote sites. As the effect of each operation applied to an application is explained by the application itself, this method is more accurate than interpreting operations based on the API provided by an operating system.

On the other hand, the above scheme heavily relies on the APIs provided by the applications, so that these APIs should satisfy the following requirements:

1) APIs should be able to intercept any operation that should be replayed at remote sites

If operation semantics is intercepted by event-generation/handler mechanism, it is obvious that any interpretable operation should be able to trigger event(s) defined by the API of a commercial application. However, the API provided by a commercial application usually only defines events for some operations that are regarded as important. For example, the operations that create/delete/update graphic objects often generate events in a graphic editing application, but the operations moving the scroll bars to change the views of a document may not. Accordingly, some operations that should be replayed at remote sites in a collaborative application may be unable to be intercepted by a single-user application's API. For instance, the operations that move the scroll bars must be intercepted in a collaborative system, if we want to implement a radar-view to enable users to know each other which part of the shared document a user is working on. However, an operation that moves a scroll bar will not generate API defined events in many single-user commercial systems. If an operation will not fire any event defined by the API provided by a commercial application, we have to intercept it using a low level API, such as Windows API, which intercepts operations according to the keyboard inputs, the position of the cursor or the states of mouse buttons. It is complex and inaccurate.

2) APIs should provide access to before and after states

To correctly intercept the semantics of an operation in concurrent environments, APIs should provide access to the document states both before and after the execution of an operation. For example, if a user changes the color of a Visio shape, *A*, from *white* to *red*, an event defined by Visio API will be triggered, which contains the information about what attribute (*color*) of which shape (*A*) is updated to what value (*red*) by the operation. This is often enough for a single-user application. However, many consistency maintenance strategies, such as OT, need to know the states of objects both before and after the executions of operations for performing consistency maintenance and achieving Any-undo in concurrent environments. The previous color (*white*) of *A* is not described in the event, so that internal data-structures have to be maintained in CoVisio to save the previous color of *A* and "old" states of other Visio objects. To solve this problem, API can raise two events for each operation, one before the operation taken effect, one after. Each event describes the value of the attribute targeted by the operation. Alternatively, the API can provide both old and new values of the updated attribute in the event triggered by the operation, so that only one event is needed.

3) APIs should be able to distinguish events generated by different user-operations

As a user mouse/keyboard operation may generate many events, an API must be able to distinguish events generated by different user-operations, which is very important when we consider undoing user-operations at remote sites. For example, when a user resizes a shape by a mouse operation in Visio, he/she may change the shape's *width*, *height*, or both of them. Suppose two events are fired in sequence reporting the *width* and *height* changes of a shape respectively. API must be able to inform CA whether these events are fired by one user-operation or two. Without knowing this information, once the *width-change* and *height-change* messages are propagated to a remote site, that site does not know whether to treat them as the effects of one user-operation or two. If they are the effects of one user-operation but treated as two, once a user presses Ctrl+Z to undo the operation at the remote site, only partial effects of the operation will be undone.

Visio API is able to distinguish events triggered by different operations. Visio API defines two events: *EnterScope* and *ExitScope*. The former will be fired when a user keyboard/mouse operation is applied to Visio user-interface or CA invokes Visio API *BeginUndoScope* method to begin the execution of a remote operation. The latter will be triggered when a user keyboard/mouse operation applied to Visio user-interface ends or when CA invokes Visio API *EndUndoScope* method to finish the execution of a remote operation. Therefore, if operations are handled in sequence at each collaborating site (this is true in most of replicated collaborative systems, including CoWord, CoPowerPoint and CoVisio, for consistency maintenance), all the events generated between a pair of *EnterScope* and *ExitScope* events are triggered by a single user-operation.

4) APIs should be able to distinguish between events generated by Do and Undo/Redo operations

An operation may generate many events. On the other hand, the same event can be generated by different operations. In a collaborative system, it is very important to distinguish between a Do-operation and an Undo/Redo-operation, a local operation and a remote operation. For example, suppose the color of a Visio object, *A*, is *red* initially. User-1 changes the color of *A* to *black*. Then user-2 changes the color of *A* back to *red* again. Once CA obtains the event reporting the effect of user-2's operation (i.e. *A* is colored to *red*), CA must also be informed whether it is the effect of a newly executed Do-operation (i.e. user-2 executes another operation which colors *A* to *red*), or just the result of an Undo-operation (i.e. user-2 undoes user-1's operation that colors *A* to *black*). This makes difference when anyone of the users undoes an operation. In the former condition, the color of *A* will be changed to *black* again. However, *A* will remain in *red* in the latter case. Therefore, once an operation is executed, API will not only inform CA the effect of the operation but also indicate whether it is a Do or an Undo/Redo operation. CA will propagate this information to remote sites so that suitable API functions can be invoked to replay the operation there.

Visio API is able to distinguish between the effects of a Do-operation and an Undo/Redo-operation. Visio application object has an *IsUndoingOrRedoing* property, which determines whether the current event handler is being called as a result of an Undo or Redo action in the application.

5) APIs should provide ways to distinguish between events resulting from user actions and those resulting from other API manipulations

Once an operation is propagated to a remote site, an API function will be invoked to replay the operation at the site. As the execution of the API function may change the attribute of a graphic object, or create/delete a graphic object, the API function may also trigger the application-defined events.

Both local operations, executed by the local user, and remote operations, replayed at a site by invoking API functions, trigger events, but each collaborating site should only propagate operations generated locally to remote sites. It is undesirable to propagate remotely generated operations, as it may result in cyclic operation propagations. Thus, APIs must be able to distinguish events generated by local and remote operations. Without this information, a CA does not know whether or not to propagate the effect of an operation, intercepted by Visio API, to remote sites.

Visio API provides functions to identify operations that fire Visio API defined events. Accordingly, we can use these functions to distinguish events generated by local users and events generated by API function invocations, which replays remote operations in CoVisio applications. To determine whether events CA received are triggered by remote operations, CoVisio uses the *BeginUndoScope* and *EndUndoScope* methods to wrap each remote operation. For example, CA will invoke the following Visio API method to begin the execution of a remote operation:

```
scopeID=visioApplication.BeginUndoScope(
"remote");
```

VisioApplication is a Visio application object. This method-invocation starts a transaction with a unique scope ID for an instance of Microsoft Visio. Accordingly, in event handlers, CoVisio uses the *IsInScope* property of the Visio application object to test whether the scope ID returned by the *BeginUndoScope* method is part of the current context. If so, the event is fired by replaying a remote operation, so that it will not be marshaled and sent to remote sites, as shown below:

```
if (visioApplication.get_IsInScope(scopeID))
{ System.Diagnostics.Debug.WriteLine("Event
generated by a remote operation"); }
```

B. API's Role in Replaying Remote Operations

Replaying operations at remote sites is another important role that APIs play in collaborative systems. In CoVisio, whatever changes a user makes to the shared Visio document via Visio user-interface at a site will be replayed on all other copies of the same document replicated at other collaborating sites. For example, once a user colors object *A* to *red* at a site, the semantics of the operation will be intercepted and marshaled into messages sent to remote sites, where the operation will be replayed to color the other replicas of *A* to *red*.

To intercept the semantics of an operation in an application, we may rely on the API provided by the operating system, if the operation cannot be intercepted by the API provided by the application. However, to replay a remote operation that manipulates an object in a commercial application, we usually can only rely on the application's API.

To replay remote operations at a site, APIs should provide functions to software developers so that whatever users can do on the user interface of an application the software developers can do by program. Obviously, if Visio API cannot provide the function to change the color of a shape, once a user changes the color of a shape in Visio GUI, the user-operation cannot be replayed at remote sites by API function invocation.

Moreover, APIs should be able to replay operations with multiple effects. For example, when a user resizes a shape by mouse operation in Visio, he/she may change both *width* and *height* of the shape. Once the effects of the user-operation are propagated to a remote site, Visio API functions will be invoked to change both the *width* and *height* of the shape replicated at the remote site. Here, it is very important to associate these effects with a single user-operation. Otherwise, when a user undoes the operation at the remote site, only partial effect of the original operation will be undone.

Visio API supports replaying operations with multiple effects. As introduced previously, Visio API provides *BeginUndoScope* and *EndUndoScope* methods to wrap multi-effect operations. Method *BeginUndoScope* starts a transaction and *EndUndoScope* ends a transaction. By replaying all the effects of a remote operation in the between of a pair of *BeginUndoScope* and *EndUndoScope* method-invocations, CoVisio ensures a multi-effect operation be replayed and undone/redone atomically at remote sites.

Two kinds of operations may be replayed at remote sites in a collaborative application, Do-operations and Undo/Redo-operations. Undo/Redo schemes are quite different between single-user and collaborative applications. Almost all commercial single-user systems only support undoing/redoing operations in sequence. For example, if the execution of operation O_2 follows O_1, O_1 cannot be undone before O_2 is undone. On the other hand, most of collaborative systems support undoing/redoing operations in arbitrary order. For instance, suppose only operation O_1 has been executed (completed at all the collaborating sites) in a collaborative system when two users concurrently execute operations at two sites. One is from site-1 to undo O_1, the other from site-2 to execute operation O_2. When the operation undoing O_1 arrives at site-2, O_2 has been executed there (i.e. O_1 is followed by O_2 at site-2). If operations can only be undone in sequence, the operation undoing O_1 cannot be executed at site-2. To enable both Do and Undo/Redo operations be replayed at remote sites, a collaborative system should be able to undo/redo operations in any order at any collaborating site.

As nowadays APIs provided by single-user commercial systems only support sequential Undo/Redo, developers may encounter problems when they implement Undo/Redo functions in a collaborative system based on the Undo/Redo API provided by a single-user commercial system. For example, in the above scenario, when the operation undoing O_1 arrives at site-2, it cannot be replayed by invoking any API Undo function, as O_2 has not been undone at site-2. One solution is to undo O_2 first then O_1. However, we cannot restore the effect of O_2 after undo O_1 by invoking API Redo function. To redo O_2, we have to redo O_1 first, as both Undo and Redo should be performed in sequence. Another solution is to do a new operation which has the same effect as undoing O_1 at site-2. For example, suppose that O_1 is to change the color of an object from *red* to *white*. Rather than undoing O_1 at site-2, we can invoke API function to execute a new operation that updates the color of the same object to *red* (i.e. reversing the effect of O_1). This solution also has serious problems. Only two operations, O_1 and O_2, are executed and O_1 is undone. If one user presses Ctrl+Z to undo the other operation, the document should return to its initial state. However, as an Undo is achieved at a remote site by doing a new operation, pressing Ctrl+Z will not bring the document back to its initial state at the remote site.

Visio Undo manager maintains Undo and Redo stacks. When a user executes an operation, the operation is put on the top of the Undo stack automatically by Visio Undo manager. When a user clicks **Undo** in Visio menu, the Visio Undo manager removes the most recently added operation from the Undo stack, and puts it on the top of Redo stack. Then the effect of the operation is undone automatically. When a user clicks **Redo** in Visio menu, the reversed procedure is preformed, an operation being moved from Redo stack to Undo stack. As Visio Undo/Redo API only allows developers to manipulate the operations on the top of Undo and Redo stacks, instead of moving any operations between Undo and Redo stacks, Visio API does not support undoing/redoing operations in arbitrary order. Accordingly, CoVisio has to abolish Visio's Undo/Redo mechanism and implement its own Undo/Redo scheme

To undo/redo operations in arbitrary order, API provided by operating system is adopted in CoVisio, so that any user

undo/redo operation is intercepted before being applied to the application. Then, Operational Transformation (OT) Any-undo scheme is used to generate an operation whose execution on the current document state will achieve the correct undo/redo effect. After that, Visio API functions will be invoked to execute the operation. In a word, with TA approach, it is the CA and GCE's responsibility to support undo/redo operations in arbitrary order. It does not rely on the SA's undo/redo API function. Please refer to [27] to find the detailed information for undoing/redoing operations in arbitrary order in collaborative systems.

C. Supporting Data Models Adaptation

It is common that the data and operation models defined by a commercial system cannot match the models required by a Generic Collaboration Engine (GCE). Accordingly, Collaboration Adapter (CA) is implemented to bridge these gaps. It is obvious that the more dissimilar the models the more complex the CA component.

One general difference between the data models defined by a single-user commercial system and a GCE is the way to address objects. In an object-oriented single-user application, such as Microsoft Visio, all the objects are addressed and accessed according to their object-references. Moreover, an object is often associated with a unique ID/name. The reference of an object can be obtained according to a unique object ID/name. Both object IDs/names and references have effects at most in the scope of a single-user application, so that they cannot address the copies of an object replicated at remote sites. On the other hand, a collaborative system should be able to address editable objects in the scope of a multi-user application, which may span several collaborating sites.

1) Supporting Global Object ID (GOID)

One well-known method adopted in collaborative systems to address objects globally is to associate the same Global Object ID (GOID) with all the replicas of the same object. For example, in a collaborative graphic editing system, when a new graphic object (not a replica of an existing object) is created, it is assigned with a GOID, which is the combination of the ID of the site the object generated from and the ID of the operation that creates that object. Then the message describing the object-generation operation will be propagated to remote sites. The GOID of the newly generated object is contained in the message. Once the message arrives at a remote site, the operation is replayed there, so that a replica of the object will be created at the remote site. Moreover, the same GOID will be used to address the replica of the object at the remote site as well. Therefore, all the replicas of the same object have the same GOID, which will not be affected by the creation and deletion of any other objects.

There are two advantages of the GOID scheme. First of all, as this method associates all the replicas of the same object with the same GOID, the GOID of an object at a site can be used directly to address the replicas of the object at remote sites. Moreover, it does not require editable objects be organized in any specific data structure. However, this method is not suitable for text editing systems, where characters are often identified according to their positions in a linear address space, as associating each copy of each character in a shared text document with a GOID is not efficient.

It is desirable that the object-accessing method provided by an API can directly match the requirement of a collaborative system, so that less work is needed to adapt Single-user Application (SA) to Generic Collaboration Engine (GCE) of a collaborative system. GOID is a well-known method adopted in collaborative systems, so that it is desirable that APIs provided by single-user commercial systems can provide methods to support GOID.

Visio API provides functions that can be extended to support GOID in concurrent environments. In addition to ID, each Visio object is also associated with a Unique ID (UID) (Visio requires that the UID of an object must be unique within the scope of a document, while the ID of a shape is unique only within the scope of a drawing page). Moreover, Visio also provides API functions for users to set UID of an object or to access an object according to its UID (A Visio object ID is read-only which is assigned by Visio automatically). If we associate the same value with the UIDs of all the replicas of the same object in a CoVisio application, an UID can be regarded as a GOID.

2) Supporting eXtended OT Data Model (XOTDM)

To facilitate consistency maintenance in collaborative systems, many strategies have defined their own sophisticated object-addressing schemes. It is desirable that APIs provided by commercial systems can also support some widely applied strategies.

Operational Transformation (OT) is an innovative and well-known consistency maintenance strategy that can be adopted in both collaborative text and graphic editing systems. The basic idea of OT is to transform (or adjust) the parameters of operations according to the effects of previously executed concurrent operations so that the transformed operations can achieve the correct effects and maintain document consistency [27], [28].

To apply OT in a wide variety of commercial applications, CoWord/CoPowerPoint proposed an eXtended OT Data Model (XOTDM), where editable objects are grouped into tree-structured hierarchical domains and the objects in the same domain are organized in a linear address space [30]. OT is implemented in CoVisio for consistency maintenance. Accordingly, CoVisio organizes Visio objects in different domains based on Visio object model. For example, each Visio application, document, or page can be regarded as a domain. A Visio application contains many documents, a document containing many drawing pages, and a page contains many shapes. CoVisio addresses objects in the same domain according to their positions in a linear address space. For instance, objects in the same drawing page are addressed according to their positions in the same Z-axis. Accordingly, a vector is used to identify a CoVisio object. For example, a vector, vp = [("application", 1), ("document", 2), ("page", 3)], refers to the graphic object which has a Z-order value "3" in the 2nd "page" of "document" "1" of a Visio "application".

XOTDM is a generic data model that can be applied to both collaborative text and graphic editing applications. However,

not all applications provide API functions supporting XOTDM. For example, in CoPowerPoint, graphic objects in the same drawing area are identified according to their Z-order values. Fortunately, Microsoft PowerPoint provides API for users to access objects according to their positions in the Z-order stack. Therefore, OT's object-addressing scheme can be directly supported by PowerPoint API. On the other hand, Visio API only provides functions to access objects using object-references, so that a Visio shape cannot be addressed according to its Z-order value in a drawing page. As Visio does not provide functions to access an object according to its position in a linear address space, XOTDM cannot be supported directly by Visio API.

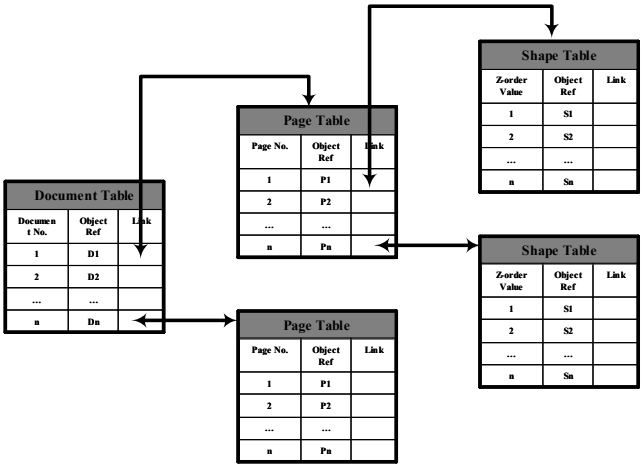

Figure 3. Layered tables for CoVisio object-IDs and Visio object-references mapping

To solve this problem, CoVisio maintains a set of layered tables to map between CoVisio shapes' hierarchical IDs (vector) and Visio object references, as shown in figure 3. Tables in different levels map between Visio object-references and CoVisio object-IDs in different domains: The first level tables are to map document-numbers and document-references, the second level tables to map page-numbers and page-references, the last level tables to map shapes' Z-order-values and shape-references. All of these tables have the same structure. Each table entry contains three fields: the position of an object in a linear space, the object's reference, and the link to the table mapping references and IDs of all the objects in the domain represented by the object. For example, in a CoVisio application, OT component uses vector, vp = [("application", 1), ("document", 2), ("page", 3)] to identify the graphic object which has a Z-order value "3" in the 2nd "page" of "document" "1" of a Visio "application". Once this hierarchical CoVisio object ID is passed to CA, CA uses "1" to index into document-table of the application. Following the *link* field of the document-table entry, CA can find the page-table that maps between CoVisio page-numbers and Visio page-references of all the pages in document "1". Then CA uses "2" to index into the page-table, and from the *link* field of the entry it can find the shape-table that maps between the CoVisio shapes' Z-order values and Visio object-references of all the Visio shapes in the drawing page "2" of document "1". Accordingly, CA uses "3" to index into the shape-table. From the *object-reference* field of the entry, it can obtain the Visio object-reference of the shape that has a Z-order value "3" in that drawing page. This object-reference will be passed to Visio API functions to access that Visio object.

Maintaining the layered tables, CA component of CoVisio is able to map between Visio object references and OT's XOTDM object IDs. However, this method consumes extra storage space and CPU time. So why was the GOID scheme is not adopted in CoVisio? If GOID is adopted, OT, which requires XOTDM, cannot be applied. Devising and implementing a new consistency maintenance strategy could be much more complicated than maintaining some internal data-structures to bridge the gaps between OT and Visio data models. Moreover, there are other advantages of applying OT in collaborative graph editing systems, which can be found in [30]. Obviously, if the API provided by a commercial application organizes objects based on XOTDM, which is a generic data model and is able to address objects in both text and graphic editing systems, leveraging the commercial system for multi-user collaboration can be more efficient.

V. CONCLUSION

With the increasing importance of using computers to support collaborative work, it is desirable to leverage commercial single-user systems for multi-user collaboration. APIs enable software developers to access and manipulate objects in a commercial application without knowing the implementation details of the application, which makes it possible for collaborative system developers to extend single-user commercial systems for multi-user collaboration. However, APIs provided by most of commercial systems are based on single-user applications. They may be unable to intercept and replay operations correctly in collaborative environments. Moreover, they may define application-specific data and operation models that cannot satisfy the requirements of a collaboration system. It is the API provided by a single-user commercial system that determines the complexity and workload to transparently leverage the single-user system for multi-user collaboration.

In this paper, we provided API design recommendations for single-user applications. We present issues that must be addressed by API designers to support intercepting and replaying operations in collaborative systems in detail. Moreover, we discussed the issues of efficient support for data models adaptation. These recommendations are obtained from our lessons learnt from developing CoVisio system. They are helpful to both API designers and collaborative system designers. API designers can apply these recommendations in API designs to support collaboration. On the other hand, based on these recommendations, a collaborative system designer knows whether a single-user commercial system can be transparently extended for collaboration according to its API functions.

Our API design recommendations are not limited to leverage single-user commercial systems for multi-user collaboration. Even system developers building collaborative systems from scratch, it would be a good approach to separate the single-user/editor part from collaborative/network parts.

Therefore, the API design recommendations can be used as a guideline for defining the interactions between the single-user part and collaborative parts of a concurrent system.

There are other API design-issues that should be addressed to provide better collaboration support, such as concurrency control and event propagation, etc. These issues are API implementation related, which are currently being investigated and will be reported in our future publications.

Over the last fifteen years, real-time collaborative systems have moved from being prototypes in laboratories to becoming usable commercial systems and also freeware. With the investigation of API design issues, we hope to make real-time collaboration even much easier to build and use.

REFERENCES

[1] J. Begole, M. Rosson, and C. Shaffer, "Flexible collaboration transparency: supporting worker independence in replicated application sharing systems". ACM Transactions on Computer-Human Interaction, Vol.6, No. 2, pp. 95−132. Jun. 1999.

[2] J. Begole, R.B. Smith, C.A. Struble, and C.A. Shaffer, "Resource sharing for replicated synchronous groupware", *IEEE/ACM Transactions on Networking*. Vol. 9, No. 6, pp. 833-843, Dec. 2001.

[3] J.D. Campbell, "Multi-user collaborative visual program development", In *Proceedings of the IEEE Symposia on Human Centric Computing Languages and Environments*, Arlington, VA, pp. 122-130, 2002.

[4] J. D. Campbell, "Interaction in collaborative computer supported diagram development", *Computers in Human* Behavior Vol. 20, No.2, pp.289-310, 2004.

[5] J.D. Campbell, "Coordination for multi-person visual program development", *Journal of Visual Languages and Computing,* Vol. 17, pp.46-77, 2005.

[6] P. Dourish, "Developing a reflective model of collaborative systems", *ACM Transactions on Computer-Human Interaction*, Vol. 2, No.1, Mar. 1995.

[7] P. Dourish, "Consistency guarantees: exploiting application semantics for consistency management in a collaborative tookit", *In Proceedings of the ACM Conference on Computer Supported Cooperative Work, ACM*, New York, pp.268-277, 1996.

[8] Ellis, C. A. and Gibbs, S. J. Concurrency control in groupware systems, In *Proceeding of ACM SIGMOD Conference on Management of Data*, (1989), 399-407.

[9] C.A. Ellis, S.J. Gibbs, and G.L. Rein, "Groupware: some issues and experiences", *CACM*, Vol.34, No.1, pp.39-58, Jan. 1991.

[10] D. Garfinkel, B. Welti, and T. Yip, "HP SharedX: A tool for real-time collaboration". HP Journal 45(2), pp. 23−36, Apr. 1994.

[11] S. Greenberg, and D. Marwood, "Real time groupware as a distributed system: concurrency control and its effect on the interface", In *Proceeding of ACM Conference on Computer Supported Cooperative Work*, pp. 207-217, Nov. 1994.

[12] M.J. Knister, and A. Prakash, "DistEdit: A distributed toolkit for supporting multiple group editors", In *Proceedings of ACM Conference on Computer-Supported Cooperative Work*, pp.343-355, 1990.

[13] D. Li, and R. Li, "Transparent sharing and interoperation of heterogeneous single-user applications", In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pp.246-255, 2002.

[14] D. Li, and R. Li, "Preserving operation effects relation in group editors", In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pp.457-466, 2004.

[15] D. Li, and J. Lu, "Performance & architecture: A lightweight approach to transparent sharing of familiar single-user editors", In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pp.139-148, 2006.

[16] D. Li, and J. Patrao, "Demonstrational customization of a shared whiteboard to support user-defined semantic relationships among objects", In *Proceedings of the ACM GROUP'01 Conference*, Boulder, Colorado, USA, pp.97-106, 2001.

[17] K. Lin, D. Chen, C. Sun, and R.G. Dromey, "Maintaining constraints in collaborative graphic systems: the CoGSE approach", In *Proceedings of the 9th European Conference on Computer- Supported Cooperative Work (ECSCW05),* 2005.

[18] K. Lin, D. Chen, C. Sun, and R.G. Dromey, "Maintaining multi-way dataflow constraints in collaborative systems", In *Proceedings of the IEEE 2005 International Conference in Collaborative Computing: Networking, Applications and Worksharing*, San Jose, CA, USA, Dec. 2005.

[19] K. Lin, D. Chen, C. Sun, and R.G. Dromey, "Multi-way dataflow constraint propagation in real-time collaborative systems", In *Proceedings of the IEEE 2006 International Conference in Collaborative Computing: Networking, Applications and Worksharing*, Atlanta, Georgia, USA, Nov. 2006.

[20] A. MacLean, K. Carter, L. Lovstrand, and T. Moran, "User-tailorable systems: Pressing the issues with buttons", In *Proceedings of ACM CHI'90 Conference*, 1990.

[21] E. Monfroy, and C. Castro, "Basic components for constraint solver co-operations", In *Proceedings of SAC*, 2003.

[22] B.A. Myers, "Graphical techniques in a spreadsheet for specifying user interfaces", In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems,* User Interface Management Systems, pp.243-249, 1991.

[23] C. Palmer, and G. Cormak, "Operation transforms for a distributed shared spreadsheet", In *Proceeding of the ACM Conference on Computer-Supported Cooperative Work*, pp.69-78, 1998.

[24] K. Rodham, and D. Olsen, "Smart telepointers: maintaining telepointer consistency in the presence of user interface customization", *ACM Transactions on Graphics,* Vol. 13, No. 3, pp.300-370, Jul. 1994.

[25] M. Roseman, and S. Greenberg, "Building real-time groupware with groupkit, a groupware tookit", *ACM Transactions on Computer-Human Interaction*, Vol. 3, No.1, pp.66-106, Mar. 1996.

[26] S. Sarin, and I. Greif, "Computer-based real-time conferencing systems", *IEEE Computer*, Vol. 18, No. 10, pp.33-45, Oct. 1982.

[27] C. Sun, "Undo as concurrent inverse in group editors", *ACM Transactions on Computer-human Interaction,* Vol. 9, No.4, pp.309-361, Dec. 2002.

[28] C. Sun, and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems", *ACM Transactions on Computer-Human Interaction*, Vol. 9, No.1, pp.1-41, Mar. 2002.

[29] C. Sun, X. Jia, Y. Zhang,Y. Yang, and D. Chen, "Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems", *ACM Transactions on Computer-human Interaction*, Vol. 5, No.1, pp.68-108, Mar. 1998.

[30] C. Sun, Q. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent adaptation of single-user applications for multi-user real-time collaboration", A*CM Transactions on Computer-Human Interaction (TOCHI*), Vol. 13, No.4, pp.531-582, Dec, 2006.

[31] D. Sun, Q. Xia, C. Sun, and D. Chen, "Operational transformation for collaborative word processing", In *Proceeding of the ACM Conference on CSCW*, Chicago, USA, Nov. 2004.

[32] N. Wilde, and C. Lewis, "Spreadsheet-based interactive graphics: from prototype to tool", In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, Application Areas, pp.153-159, 1990.

[33] Q. Xia, D. Sun, C. Sun, D. Chen, and H. Shen, "Leveraging single-user applications for multi-user collaboration: the CoWord approach", In *Proceeding of the ACM Conference on CSCW*, Chicago, USA, pp.162-171, Nov. 2004.