Object-Associated Telepointer for Real-time Collaborative Document Editing Systems

Object-Associated Telepointer for Real-Time Collaborative Document Editing Systems

Steven Xia
Griffith University
Brisbane, QLD 4111, Australia

David Sun
University of California at Berkeley
Berkeley, CA, USA

Chengzheng Sun
Nanyang Technological University
Singapore 639798

David Chen
Griffith University
Brisbane, QLD 4111, Australia

Abstract

Telepointers are a real-time groupware interface feature to indicate where other users are pointing. None of existing telepointer techniques, however, is capable of tracking the reference object and preserving the relative position in the face of dynamic content and view changes in real-time collaborative document editing systems. In this paper, we report a novel Object-Associated Telepointer (OAT) technique with the following features in the face of dynamic content and view changes: (1) relocating telepointers to track the reference object, (2) preserving the position relative to the reference object, and (3) providing feedback of the telepointer relocation to the local user with a virtual local cursor. The key technique for supporting OAT is to extend the operational transformation technique with a generic Refer operation to support adjusting the reference object position. The OAT technique has been implemented in the CoWord system based on the transparent adaptation approach so that it can provide the OAT support to a range of real-time collaborative applications.

1. Introduction

Telepointers are avatars of local users' mouse cursors displayed on remote participants' screens in real-time groupware systems. As an important groupware interface element, telepointers are able to provide a variety of group awareness information including *presence, location and activity*. In addition, telepointers can act as a communication channel by conveying *gestural* messages [9][10]. These features make telepointers a powerful means for providing users with a collaboration context, helping users coordinate the group work and improving groupware usability [8][11].

We are particularly interested in the telepointer techniques in real-time collaborative document editing systems which allow multiple users to edit any objects in the shared document at any time. In these systems, awareness of other participants' presence, location, and activity are important for collaboration and coordination, so telepointers can be a useful tool.

For users to obtain meaningful location and activity information from a telepointer, it is important that the telepointer points to precisely the same *reference object* as the local cursor does. This reference object is usually the one existing at the local cursor position on the user interface. However, in different situations, the types of the reference object can be different. For example, the reference object can be an interface widget (e.g. a button) in the shared window or a content object in a shared document (e.g. a character in a text widget).

In existing telepointer techniques, the reference object pointed to by a telepointer is *static* in the sense that the object identifier does not change [6][14]. For example, while referring to a window widget, the telepointer is associated with the target widget identifier, which never changes. While referring to a character within a text viewing widget, the telepointer is associated with a constant widget identifier plus a constant index of the character in the text buffer. The invariable object identifiers ensure the correctness of the static reference scheme in a range of groupware systems.

Unfortunately, the static reference scheme does not work in real-time collaborative document editing systems. This is because in such systems, users can edit any objects in the shared document at any time. As a result, positional references of content objects are subject to dynamic changes. These changes may cause problems to telepointing under two circumstances. First, when a telepointer is about to be relocated to a new reference object in response to the local cursor movement, the reference object may have been moved or changed by editing operations concurrent with the cursor movement, causing the telepointer to point to an incorrect object. Second, after a telepointer is relocated to a reference object, the reference object may be moved by subsequent editing operations or view changes, which may also cause the telepointer to point to an incorrect object.

1-4244-0030-9/05/$20.00 ©2005 IEEE

The root of these problems is that the static reference scheme assumes a one-to-one and static correspondence between the object identifier and the object itself and simply associates telepointers with object identifiers. If the correspondence between the object and its identifier changes dynamically, telepointers may fail to point to the correct object.

To address this object-association problem in dynamic and concurrent shared workspaces such as real-time collaborative document editing systems, we propose an *Object-Associated Telepointer* (OAT) technique in this paper. The basic idea is to adjust telepointer reference parameters by means of *Operational Transformation* (OT) [17] so that a telepointer is always associated with a reference object in the face of dynamic content and view changes. The OAT technique is devised in the *Transparent Adaptation* (TA) [21] framework, which is able to convert single-user commercial applications into collaborative versions without changing the source code. This framework provides the necessary technical infrastructure for implementing the OAT technique and facilitates the application of the OAT technique to a range of existing commercial single-user applications.

The rest of this paper is organized as follows. First, background knowledge about OT and TA is briefly introduced. Afterwards, problems with existing telepointer techniques are discussed as the motivation of this work. Next, the effects that the OAT technique should achieve are discussed. Then technical issues related to achieving OAT effects are discussed and OT-based solutions are proposed. Furthermore, issues related to implementing OAT in the CoWord system are discussed. Finally, contributions and future work are summarized.

2. Background on OT and TA

2.1 Basics of the OT Technique

OT was originally designed to support multiple users to insert and delete characters in replicated text documents concurrently [4][17]. Due to its unique capability in achieving system consistency without imposing any restrictions on users, OT has become the choice of consistency maintenance and group undo technique for many collaborative editing systems [1][12][19].

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: O1 = Insert [0, "x"] (to insert character "x" at position "0"), and O2 = Delete [3, "c"] (to delete the character "c" at position "3") generated by two users at collaborating sites 1 and 2, respectively. Suppose the two operations are executed in the order of O1 and O2 (at site 1). After executing O1, the document becomes "xabc". To execute O2 after O1, O2 must be transformed against O1 to become: O2' = Delete [4, "c"], whose positional parameter is incremented by one due to the insertion of one character "x" by O1. Executing O2' on "1abc" shall delete the correct character "c" and the document becomes "xab". However, if O2 is executed without transformation, then it shall incorrectly delete character "b", rather than "c".

In summary, the basic idea of OT is to transform an editing operation (e.g. O1) defined on a previous document state according to the effects of executed concurrent operations (e.g. O2), so that the transformed operation (e.g. O2') can achieve the correct effect in the current document state.

2.2 Basics of the TA Approach

TA is an innovative approach to converting exiting or new single-user applications for multi-user real-time collaboration, without changing the source code of the original application [21]. The TA approach is based on a replicated system architecture where the shared single-user application is replicated at all collaborating sites, the use of the single-user application's API (Application Programming Interface) to intercept and replay the user's interactions with the shared application, and the use of the OT technique to manipulate the intercepted user operations for supporting responsive and unconstrained (i.e. concurrent and free) multi-user interactions with the shared application. The central idea of the TA approach is to adapt the data address and operation models of the shared application's API to that of the OT technique.

More precisely, the TA approach can be described by a reference model, as shown in Figure 1. This reference model consists of three components: *Single-user Application* (SA), *Collaboration Adaptor* (CA), and *Generic Collaboration Engine* (GCE). The main functionalities of these components are sketched below.

The SA component provides conventional single-user interface features and functionalities. This component can be either an existing commercial off-the-shelf single-user application, or a new single-user functionality component in a multi-user collaborative system, but this component itself has no knowledge about multi-user collaboration.

The CA component provides application-specific collaboration capabilities and plays a central role in adapting the SA for collaboration. This component has the knowledge of the SA API but not its internals. At the center of this component is the module of *Adapted Operation* (AO), which represents the SA functionalities exposed by the API. The AO can be generated by the *Local Operation Handler* (LOH) module by intercepting

local user's interactions, or received by the *Remote Operation Handler* (ROH) module from remote users. With the AO residing between the API and OT, the task of adaptation between the API and OT is decomposed into two modules:

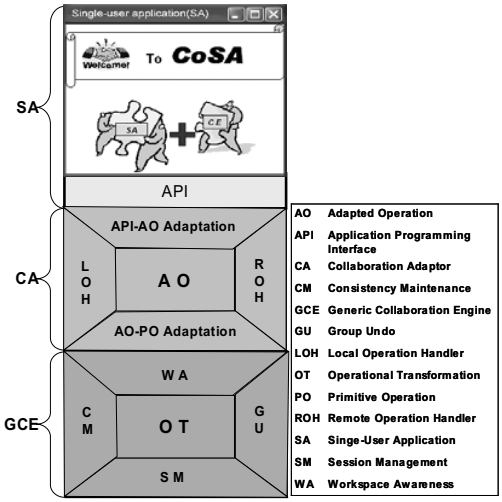

Figure 1: The TA reference model.

1. The *API-AO Adaptation* module is responsible for bridging the semantic gap between the API and the AO so that the AO can be correctly replayed on the SA.
2. The *AO-PO Adaptation* module is responsible for mapping between the AO and OT-supported *Primitive Operation* (PO) so that the underlying OT technique can be used to ensure the correctness of the AO parameters in the presence of concurrency.

The GCE component provides application-independent collaboration capabilities. This component has no knowledge of the single-user application functionality and therefore can be used in adapting different applications. This component encapsulates a package of collaboration supporting techniques, including *Consistency Maintenance* (CM), *Group Undo* (GU), *Workspace Awareness* (WA), and *Session Management* (SM), etc. OT is at the core of this component for supporting consistency maintenance, user-initiated undo, and workspace awareness (e.g. the telepointer discussed in this paper) in a collaborative environment. For comprehensive discussions of the techniques encapsulated in the OT module, the reader is referred to [19][17][18][20].

3. Related Work

The telepointer has a history almost as long as the pointing device (mouse) [5]. The telepointer's reference scheme has been evolving over the years.

3.1 Window Coordinate-Associated Telepointers

In early application sharing and teleconference systems like Colab [16] and MMConf [3], a telepointer is associated with the coordinates in the shared window space. In these systems, a strict WYSIWIS view mode is adopted. As a result, all users have exactly the same view of the shared window. This ensures that each object is placed at the same position in the shared window, and the same coordinates point to the same object at all sites, so the window coordinates are sufficient for a telepointer to locate any objects in the shared window.

3.2 Widget-Associated Telepointers

In a relaxed WYSIWIS view mode, a shared window can have different layout among participating sites. To accommodate the view difference, techniques associating telepointers with window widgets have been proposed, including Smart Telepointer [14] and GroupKit [15]. With these techniques, a telepointer is associated with identifiers of a window widget, and provided with the relative position inside the widget space. For example, in Smart Telepointer, the telepointer's reference parameters include a path in the widget tree from the root to the leaf-level widget which contains the telepointer and relative position information within the leaf-level widget space.

3.3 Object Position-Associated Telepointers

Some widgets may have internal structures or contents (e.g. a text editor or an HTML viewer). In a relaxed WYSIWIS view mode, the internal content may be formatted and displayed differently due to different view customization among collaborating sites. For such widgets, the widget association is not enough. Smart Telepointer associates a telepointer with the content object position by attaching the index of the reference object (e.g. the character index in a text buffer) in the telepointer reference parameters, so that the telepointer can point to the same content object as the local cursor does. This technique is also adopted in GroupWeb [7] and Flexible JAMM [1].

Although these telepointer techniques work well in their own environments and could achieve the effects they were designed for, they are not applicable for telepointing content objects in real-time collaborative document editing systems due to dynamic content and view changes. To solve this problem, an Object-Associated Telepointer (OAT) technique is proposed in this paper. Unlike existing telepointer techniques, the OAT technique really associates a telepointer with the reference object rather than its position. It is able to correctly point to the reference object in the face of

dynamic content and view changes caused by both concurrent and sequential editing operations. In particular, it is able to achieve the object-associated effects which will be discussed in the next section.

4. Object-Associated Effects for Telepointing

In this section, we shall discuss the object-associated effects that the OAT technique should achieve.

4.1 Positional Reference Adjusting (PRA) Effect

The telepointer identifies the reference object by its positional reference in the document. The telepointer should be able to adjust its positional reference in order to track the reference object in the face of dynamic content changes caused by editing operations. Examples of the PRA effect are shown in Figure 2. At the initial state (Figure 2-(a)), the telepointer is pointing to the character "p" at position 4. After the execution of an insert or delete operation, the positional reference of the object may be changed. To achieve the PRA effect, the telepointer positional reference must be adjusted so that it still points to the character "p", as shown in Figure 2-(b) and (c). It should be pointed out that editing operations could be generated concurrently with or sequentially after a telepointer moving operation, the PRA effect must be achieved under all circumstances.

Figure 2: The PRA effect (the telepointer tracks the reference character "p"). (a) The initial state; (b) The state after executing an insert; and (c) The state after executing a delete.

4.2 Relative Position-Preserving (RPP) Effect

The telepointer's position relative to the reference object should be preserved in the face of dynamic changes to the document. An example of the RPP effect is shown in Figure 3. At the initial state (Figure 3-(a)), the telepointer is pointing at the center of the picture. After the execution of a *size-updating* operation, the picture is resized to 50% of the original size. To achieve the RPP effect, the telepointer position must be adjusted to accommodate the effect of the updating operation on the object so that it still points to the center of the picture (Figure 3-(b)).

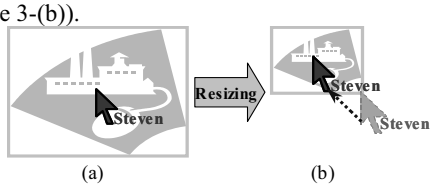

Figure 3: The RPP effect. (a) The initial state; (b) The state after executing a resize operation: the telepointer remains inside the picture.

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

An example is shown in Figure 4. In the initial state (Figure 4-(a)), the telepointer is in the blank area near the picture. After the picture is resized, the telepointer is relocated accordingly so that it still points at the same position relative to the reference object (Figure 4-(b)).

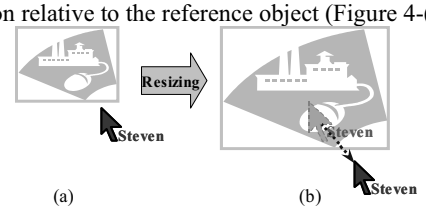

Figure 4: The RPP effect when the telepointer is in a blank area. (a) The initial state; (b) The state after executing a resize operation on the picture: the telepointer remains outside the picture.

4.3 The Virtual Local Cursor

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

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

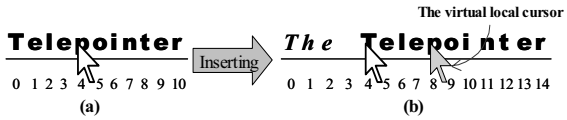

Figure 5: The virtual local cursor for tracking the reference object. (a) The initial state; (b) The state after the reference object is pushed to the right, the virtual local cursor follows the reference object, but the real local cursor is not affected.

Consider the example shown in Figure 2. When the string "The " is inserted, the telepointer is relocated to track the character "p" (Figure 2-(b)). What happens at user Steven's local site at the same time is shown in Figure 5. In the initial state (Figure 5-(a)), the local cursor is pointing at the character "p" at position 4. After the string is inserted, the virtual local cursor tracks the reference object (Figure 5-(b)), but the real local cursor is not affected.

4.4 Object-Associated Telepointing Operation Definition

The object-associated telepointing operation is defined as follows:

Cursor_move [ReferenceAddr, <RelativeX, RelativeY> ...]: to move the cursor to the position specified by the parameters.

In an application supporting OAT, the *Cursor_move* operation is generated from the local site when the local cursor moves and propagated to remote sites to relocate telepointers accordingly.

The *ReferenceAddr* parameter is a linear index to address the reference object in the document. Although this linear addressing scheme is modeled after plain text editors, graphs and other complex objects can also be mapped to a linear addressing space and be addressed with linear indices [21].

The pair <*RelativeX, RelativeY*> represents the cursor position relative to the reference object. The relative position is measured as ratios to the reference object size, rather than absolute pixel numbers. Theoretically, any point inside the reference object can be chosen as the reference point as long as all collaborating sites make the same choice. In this paper, we take the left top corner of the reference object as the reference point.

When applying the OAT technique in a concrete application, the *Cursor_move* operation may need additional parameters, such as site identifiers, document identifiers and window identifiers.

5. Achieving the Object-Associated Effects

In this section, we shall discuss technical problems and solutions related to achieving the object-associated effects.

5.1 Achieving the PRA Effect

Like editing operations (e.g. *Insert*, *Delete*), the *Cursor_move* operation also refers to the reference object with linear address. Therefore, inconsistency problems including *Causality Violation* and *Intention Violation* [19] could also occur to the *Cursor_move* operations.

Generally, the solution to the Causality Violation problem is to timestamp each operation based on vector logic clocks [2][13], and to execute an operation only when all operations causally before it have been executed. The solution to the Intention Violation problem is to transform each operation with OT before executing (for details of the Causality Violation and Intention Violation problems and solutions, the reader is referred to [19]).

However, existing OT techniques do not support transforming the *Cursor_move* operation. Here we propose an approach to supporting the transformation of the *Cursor_move* operation by extending the OT technique.

Rather than directly transforming the *Cursor_move* operation, we transform it by means of a primitive operation *Refer*. The *Refer* operation is defined as follows:

Refer[pos]: to refer to the object at the position *pos*.

The *pos* parameter indicates the address of the reference object in the document. When converting a *Cursor_move* operation into a *Refer* operation, the pos parameter equals to the *ReferenceAddr* parameter. In all *Cursor_move* parameters, only *ReferenceAddr* is included in the *Refer* operation because only this parameter may be affected by concurrent operations and is relevant to OT.

With the mediation of the *Refer* operation, transformation of a Cursor_move operation is done in the following steps:

1. The *Cursor_move* operation is converted into a Refer operation.
2. The *Refer operation* is transformed with OT.
3. The *ReferenceAddr* parameter of the *Cursor_move* operation is adjusted according to the pos parameter of the transformed Refer operation.

In this way, the *Cursor_move* is effectively transformed against concurrent operations.

The major reason for translating the *Cursor_move* operation into a *Refer* operation for OT processing is that *Refer* is more generic in the sense that, in addition to representing object reference for telepointing, it can also be used to represent other operations that only refer to objects in the document without generating any modification.

The OT technique consists of two separate layers [19]: the high-level transformation control layer and low-level transformation function layer. The transformation control layer determines which operations are transformed; the transformation function layer determines how the operations are transformed. To add a new operation into an existing OT technique, we only need to define transformation functions for this new operation while keeping the high-level control algorithm unchanged.

There are two kinds of transformation functions: *inclusive transformation* (IT) and *exclusive transformation* (ET) functions [19]. IT transforms operation Oa against operation Ob in such a way that the impact of Ob is effectively included; ET transforms Oa against Ob in such a way that the impact of Ob is effectively excluded.

For the *Refer* operation, ET functions are not needed, because *Refer* does not have effects on other operations and hence is not saved in the HB. For the same reason, IT functions that transform other operations against the *Refer* operation are not needed either. Therefore, we only need to define IT functions for transforming *Refer* against other operations.

```
IT_RI(Or, Oi)
{
   if (Oi.pos <= Or.pos)
      Or.pos = Or.pos + Oi.len;
   return Or;
}

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

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

Figure 6: IT functions for the Refer operation.

The OT technique supports three primitive operations, which are *Insert*, *Delete* and *Update* [20]. IT functions transforming Refer against these operations are shown in Figure 6.

In the function description, we use the dot notion to refer to a parameter of an operation. For example, we use *Oa.len* to refer to the *len* (length) parameter of operation Oa and use *Oa.pos* to refer to the *pos* (position) parameter of the Oa.

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

Transforming a *Refer* against a *Delete* (IT_RD) is more complex. If the range of the *Delete* operation is completely to the left of the *Refer's* position, then the *Refer's* position is shifted to the left by the *Delete's* length, because the reference object is pulled to the left. If the *Delete's* range covers the *Refer's* position, then the position of the *Refer* is set to the position of the *Delete*, because the original reference object is deleted by the *Delete* operation and the object at *Od.pos* becomes the new reference object. Finally, if the *Delete's* position is to the right of the *Refer's* position, then the *Refer's* position parameter is not changed.

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

5.2 Achieving the RPP Effect

The RPP effect can be achieved by making use of the relative ratio position parameters of the *Cursor_move* operation.

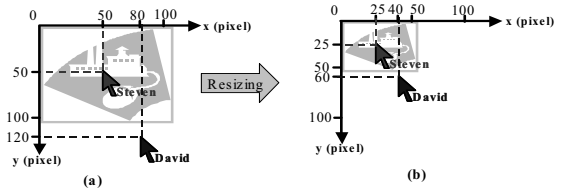

Figure 7: A scenario of achieving the RPP effect with the relative ratio position parameters. (a) The initial state; (b) The state after resizing.

When the reference object has been found from the document, coordinates of the new telepointer position can be calculated by using its current status (position and size) and the relative ratio position parameters. An example is shown in Figure 7.

In the initial state (Figure 7-(a)), the reference object occupies an area of 100 * 100 pixels. User Steven's telepointer is at the centre of the reference object and the relative ratio position is <0.5, 0.5>, corresponding to the relative pixel position <50, 50>. User David's telepointer is outside the reference object and the relative ratio position is <0.8, 1.2>, corresponding to the relative pixel position <80, 120>. After the reference object is resized to half of its original size (Figure 7-(b)), which is 50 * 50 pixels, positions of the two telepointers are recalculated. Based on the new size of the reference object and the relative ratio positions, the relative pixel position of user Steven's telepointer is changed to <25, 25>, so that it is still at the center of the reference object; the relative pixel position of user David's telepointer is changed to <40, 60>, so that it is still at the same position relative to the reference object.

5.3 Supporting the Virtual Local Cursor

To provide the local user with feedback of remote telepointer relocation, the local cursor is accompanied with a virtual local cursor. Normally, the virtual local cursor is not visible since it points at exactly the same position as the real local cursor does. When remote telepointers are relocated to track the moved reference object, the virtual local cursor needs to be relocated accordingly and then points to a different position from the real local cursor. Only in this circumstance, the virtual local cursor appears.

To keep the virtual local cursor consistent with remote telepointers, the PRA and RPP effects should also be achieved while relocating the virtual local cursor.

When the user moves the local cursor, the virtual cursor shall disappear. While moving the local cursor, the user may want to point to another object. In this case the new reference object should be identified and associated with the local cursor and telepointers, and remote telepointers are relocated accordingly. The user may also want to move the local cursor to point to the original reference object. With the virtual cursor associated with the original reference object, it is much easier for the user to find this moved object from the document. In this case, there is no need to relocate remote telepointers.

5.4 Impacts of Subsequent Editing Operations

Apart from concurrent editing operations, sequential editing operations executed after a *Cursor_move* operation may also change the *position* or *size* of the reference object and hence invalidate the association between the telepointer and the reference object.

Editing operations executed at any address could affect the position or size of the reference object. First, operations targeting on a reference object could directly change its position or size attribute. Second, operations executed before a reference object could change its linear address in the document and then affect its position (see Figure 2-(b) and (c)). Third, operations executed after the reference object could change the layout of the document view, and the reference object position may also be affected.

To solve these problems, the following telepointer relocation scheme for accommodating changes caused by subsequent editing operations is needed.

1. Addresses of all reference objects are adjusted to accommodate the effect of the editing operation.
2. New positions of all telepointers are recalculated based on the new status of reference objects.
3. Telepointers are moved to new positions if necessary.

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

6. Preserving Object-Associated Effects in the Face of View Changes

Telepointers are displayed in a layer different from the document view window. Therefore, view changes (e.g. scrolling up and down, zooming in and out) could also affect the association between telepointers and reference objects. To preserve the object-associated effects in the face of view changes, telepointers and the virtual local cursor need to be relocated as well. An example is shown in Figure 8.

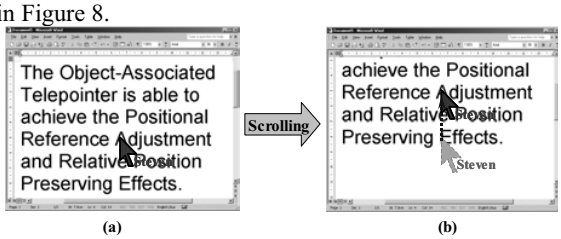

Figure 8: A scenario for preserving the object-associated effects in the face of view change. (a) The initial state; (b) The state after scrolling.

In the initial state (Figure 8-(a)), the telepointer is pointing at the center of the character "A". After the document view is scrolled up, the position of the reference object, character "A" is moved. To preserve the PRA and RPP effects, the telepointer needs to be moved up as well so that it still points at the center of the character "A" (Figure 8-(b)).

Like editing operations, view changes could also be concurrent with or sequential to the *Cursor_move* operation. Therefore, both situations should be considered in the solution.

The effect of the concurrent view change is accommodated by the telepointer position calculation. When a *Cursor_move* operation is executed at a remote site, the current status (after the view change) of the reference object is used to calculate the telepointer position. In this way, the telepointer position has taken the effect of view changes into account.

View changes affect the position and size of the reference object without changing their internal state in the document. Therefore, when a subsequent view change occurs, addresses of reference objects do not need to be adjusted. Only the telepointer position recalculation and moving are needed. In other words, the telepointer relocation scheme for subsequent view changes only

includes the second and third steps of the telepointer relocation scheme for subsequent editing operations.

For the example shown in Figure 8, after the view has been scrolled up, the new status of the reference object the character "A" is obtained. The new position of the telepointer is calculated based on the position and size of the reference object, and the relative position parameters of the telepointer. Finally, the telepointer is moved to the new position.

7. Implementing OAT in CoWord

We have implemented the OAT technique in the CoWord system (http://reduce.qpsf.edu.au/coword), which has converted MS Word into a real-time collaborative word processor with the TA approach, as shown in Figure 1. The main benefits of implementing the OAT technique in the TA framework include (1) the availability of GCE for transforming the *Cursor_move* operation with OT, (2) the availability of the SA's API for querying and updating document object information, and (3) the ability to reuse the OAT technique in a range of existing off-the-shelf commercial single-user applications.

7.1 API Support of MS Word

With the OAT technique, collaborating sites communicate the telepointer information by means of the *Cursor_move* operation. While implementing the OAT technique in a TA-based system, the SA's API should provide adequate functionality for the generation and execution of the *Cursor_move* operation.

MS Word provides an object-oriented API (http://msdn.microsoft.com) for accessing objects in a document. The document is represented with a *Document* object, and content objects in a document can be accessed by a *Range* object, which represents a linear sequence of objects. A document can be displayed in multiple windows. Each window is represented with a *Window* object. Methods and properties of these objects provide the necessary information for the generation and execution of the *Cursor_move* operation.

To generate a *Cursor_move* operation at the local site, the address, screen position and size of the reference object are needed to determine the operation parameters. The reference object can be identified by passing the local cursor position to the *RangeFromPoint* method of the *Window* object corresponding to the document window containing the local cursor. This method returns a *Range* object containing the reference object based on the cursor position. The screen position of the reference object can be obtained by passing the *Range* object into the *GetPoint* method of the *Window* object. Properties of the *Range* object can provide the size and address of the reference object.

To execute a *Cursor_move* operation at a remote site, the screen position and size of the reference object are needed to calculate the telepointer position. The *ReferenceAddr* parameter is first used to access the reference object (contained in a *Range* object) from the *Document* object. Then, the *GetPoint* method of the Window object and attributes of the *Range* object can provide information needed to calculate the telepointer position.

7.2 The OAT Table

The main data structure for supporting OAT is an *OAT Table* (OATT). The OATT contains entries for all cursors displayed on the screen (including telepointers and the local virtual cursor). Each entry contains a *Refer* operation, a relative ratio position from the *Cursor_move* operation and an absolute screen position of the telepointer. When a *Cursor_move* operation is executed, the relative and absolute positions of the telepointer are saved in the entry for the target telepointer. When a *Cursor_move* operation is propagated, a *Refer* operation converted from the *Cursor_move*, the relative and absolute positions of the local cursor are saved in the local cursor entry. When an editing operation is executed or a view change happens, the operations and positions in all entries are updated. When a site joins a collaboration session, a new entry for the new site is created in the OATTs of all existing sites; and the new site inherits the OATT of one existing site. When a site leaves a collaboration session, its corresponding entry is removed from OATTs of all existing sites.

In the TA framework, the OATT is maintained in the WA module of the GCE (see Figure 1).

7.3 Handling the Cursor_move Operation

In the CoWord system, the *Cursor_move* operation is only used to represent movements of the local cursor, so it is defined as an AO. On the other hand, the *Refer* operation is directly supported by OT and is used to represent *Cursor_move* in OT, so it is defined as a PO.

When the local user moves the mouse cursor, the cursor movements are intercepted by the LOH module (see Figure 1). Then LOH calls the API-AO Adaptation module to get information about the reference object from the Word API. Based on these parameters, a *Cursor_move* AO is generated. Next, the virtual local cursor is hidden if it is visible. Finally the *Cursor_move* AO is propagated to remote sites by the ROH module. Meanwhile, the relative and absolute positions of the local cursor and a Refer PO converted from the

propagated *Cursor_move* are saved in the local cursor entry of the OATT.

When the *Cursor_move* AO is received by the ROH module of a remote site, it is converted to a *Refer* PO by the *AO-PO Adaptation* module and transformed by OT in GCE. Then the reference object is identified from the document and its status parameters are queried from the Word API by calling the *API-AO Adaptation* module. Next, the screen position of the telepointer is calculated based on these parameters and the relative position parameters in the *Cursor_move* operation. And then, the telepointer is moved to the new position. Finally, the new relative and absolute positions of the telepointer and the transformed *Refer* PO are saved in the OATT.

7.4 Relocating Telepointers after Editing Operations

Editing operations are generated at the local site and propagated to remote sites and applied to remote document replicas. Therefore, editing operations affect telepointers at both the local and remote sites.

After a local or remote editing operation is executed, all entries in the OATT should be recalculated to adapt the change caused by the editing operation. This is done by calling the *AdjustOATT* routine shown in Figure 9.

```
AdjustOATT (O, OATT)
{
  for (i = 0; i < OATT.entries.count; i++)
  {
    IT (OATT.entries[i].Or, O);
  }
}
```

Figure 9: The routine for adjusting the OATT.

This routine transforms the *Refer* operation saved in each entry against the new editing operation. Positions of *Refer* operations are adjusted to reflect the effect of the new editing operation. The IT invocation in this routine chooses suitable IT functions shown in Figure 6 according to the new editing operation's type.

Next, all telepointer positions should be relocated by calling the routine *RelocateAllTelepointers* shown in Figure 10. The absolute screen position of the telepointer in each entry (including the local virtual cursor entry) is recalculated for its new screen position after the execution of the new editing operation. This recalculation involves the *Refer* operation, the relative ratio position saved in OATT entries and status information of the reference object that can be obtained from the Word API. If the new screen position is different from the current one, then the current screen position is replaced and the telepointer is moved to the new position. If the new local virtual cursor's position is different from the local real cursor's position, then the virtual local cursor becomes visible and is moved to the new position.

```
RelocateAllTelepointers(OATT)
{
  for (i = 0; i < OATT.entries.count; i++)
  {
    new_screen_pos = CalculatePos(OATT.entries[i]);
    if (PositionChanged(OATT.entries[i], new_screen_pos)
    {
      SaveScreenPos(OATT.entries[i], new_screen_pos);
      if ( IsLocalCursor(OATT.entries[i]) &&
        VirtualCursorIsHidden( ) )
        ShowVirtualCursor( );
      MoveTelepointer(OATT.entries[i]);
    }
  }
}
```

Figure 10: The routine for relocating all telepointers.

Unlike editing operations, view changes are not propagated to remote sites, so they only affect telepointers locally. After a view change happens at the local site, only the *RelocateAllTelepointers* routine is invoked to calculate the new positions of cursors and to move the telepointers if necessary. This is correct because view changes do not affect addresses of reference objects.

8. Conclusion and Future Work

In this paper, we have contributed a novel object-associated telepointing technique OAT that is suitable for real-time collaborative document editing. The most important feature of the OAT technique is that it is able to track the reference object and maintain the relative position to the reference object in the face of dynamic content and view changes in collaborative document editing systems. We defined the desired OAT effects, analyzed the related consistency and relocation issues, and devised solutions based on the Operational Transformation (OT) technique. We have extended the OT technique with a new generic primitive operation *Refer*, which can be used to model various object-referencing activities from the user interface. Moreover, the OAT technique has been implemented in the CoWord system to provide OAT support to real-time collaborative word processing.

We are in the process of applying the TA approach and the OAT technique to other applications, including web page designers, graphic and image editors, and CAD/CASE tools. Apart from studying the implementation techniques for supporting the known group-awareness techniques (e.g. telepointers, radar views, etc.) in TA-based systems, we also plan to investigate the special group-awareness requirements and new supporting techniques in these systems and conduct usability study of them.

9. Reference

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

[2] Birman, K., Schiper, A., and Stephenson, P. Lightweight causal and atomic group multicast. ACM Transaction on Computer Systems, 9(3), 1993, pp. 272 – 314.

[3] Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. MMConf: An infrastructure for building shared multimedia applications. *In Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, 1990, pp. 329–342.

[4] Ellis, C. A., and Gibbs, S. J. Concurrency control in groupware systems, *In Proceedings of ACM Conference on Management of Data*, May 1989, pp. 399 – 407.

[5] Engelbart, D. and English, W. A research center for augmenting human intellect. In Proceedings of the Fall Joint Computing Conference, vol. 33, 1968, pp. 395 – 410.

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

[7] Greenberg, S., and Roseman, M. Groupweb, a www browser as real time groupware. *Conference companion on Human factors in computing systems: common ground*, 1996, pp. 271 – 272.

[8] Gutwin, C. Workspace awareness in real-time distributed groupware. PhD thesis, University of Calgary, Calgary, Canada.

[9] Gutwin, C., and Penner, R. Improving interpretation of *remote* gestures with telepointer traces. *In Proceedings of ACM Conference on Computer-Supported Cooperative Work*, November 2002, pp. 49 – 57.

[10] Gutwin, C., Dyck, J., and Burkitt, J. Using cursor prediction to smooth telepointer jitter. *In Proceedings of ACM SIGGROUP conference on supporting group work*, November 2003, pp. 294 – 301.

[11] Gutwin, C., and Greenberg, S. The effects of workspace awareness support on the usability of real-time distributed groupware. ACM Transactions on Computer-Human Interaction, 6(3), September 1999, pp. 243 – 281.

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

[13] Ranynal, M. and Singhal, M. Logical time: Capturing causality in distributed systems. IEEE Comput. 29(2), February 1996, pp. 49 –56.

[14] Rodham, K. J., and Olsen D. R. Smart telepointers: maintaining telepointer consistency in the presence of user interface customization. *ACM Transactions on Graphics*, 13(3), July 1994, pp. 300 – 307.

[15] Roseman, M., and Greenberg, S. Building real-time groupware with groupkit, a groupware toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1), March 1996, pp. 66 – 106.

[16] Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., and Suchman, L. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Communication of the ACM*, 30(1), January 1987, pp. 32 – 47.

[17] Sun, C. and Ellis, C. A. Operational transformation in real-time group editors: issues, algorithms, and achievements. *In Proceedings of the ACM Conference on Computer Supported Cooperative Work*, 1998, pp. 59 – 68.

[18] Sun, C. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction*, 9(4), December 2002, pp. 309 – 361.

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

[20] Sun, D., Xia, S., Sun, C., and Chen, D. Operational transformation for collaborative word processing, *In Proceedings of ACM Conference on Computer Supported Cooperative Work*, November 2004, pp. 437 – 446.

[21] Xia, S., Sun, D., Sun, C., and Chen, D. Leveraging single-user applications for multi-user collaboration: the CoWord approach. *In Proceedings of ACM Conference on Computer-Supported Cooperative Work*, November 2004. pp 162 – 171.