43

Object-based Linear Undo model

Chunbo Zhou

Dept. of Computer Science
Yamanashi University
4-3-11 Takeda, Kofu
Yamanashi 400, Japan
shou@cluster.esi.yamanashi.ac.jp

Atsumi Imamiya

Dept. of Computer Science
Yamanashi University
4-3-11 Takeda, Kofu
Yamanashi 400, Japan
imamiya@yamato.esi.yamanashi.ac.jp

ABSTRACT The simplicity of traditional linear undo model makes it easy to understand, easy to use and widely adopted in a lot of software products on the market. But the traditional linear undo model cannot support object-based recovery operations. In order to solve this problem, we propose the object-based linear undo model in this paper. In our model, the traditional linear recovery history is reorganized into main history and subhistories. The main history represents the sequence of interactive cycles. The subhistories record the operations on corresponding objects. Traditional linear undo mechanism is improved and applied to both main history and subhistories so that the main history can support traditional linear undo/redo operations and the subhistories can support object-based undo/redo operations. The object-based linear undo model has been implemented on the platform of TRIBASE, a UIMS system developed by SEIKO Instruments Incorporation.

KEY WORDS usability engineering, interaction design, recovery model, undo, redo

Human–Computer Interaction: INTERACT'97 S. Howard, J. Hammond & G. Lindgaard (editors)
Published by Chapman & Hall

1. INTRODUCTION

Recovery facility is very important for enhancing the usability and learnability of human-computer interaction systems. So far a lot of undo models have been proposed. All the undo models can be classified into primitive undo models or meta undo models(Yang, 1988). In a primitive undo model, all the user-issued commands including both task-oriented commands and recovery commands are regarded as primitive commands. The recovery history is the list of such primitive commands which have been issued. There is only one recovery command, undo, that reverse the effect of last user-issued command. So primitive undo model is a single level undo model and seldom studied nowadays. In a meta undo model, only task-oriented commands are recorded in the recovery history. Recovery commands such as undo, redo and skip are meta commands which work on the history of task-oriented commands. The meta commands are never recorded in the history.

Furthermore, meta undo models can be classified into linear undo models or nonlinear undo models (Berlage, 1994 Yang, 1988). In a linear undo model, undo and redo commands can only work on the history by the order that task-oriented commands are recorded. A nonlinear undo model allows the user to undo or redo an arbitrary command in the history. Typical nonlinear undo models include script model (Archer, 1984), US&R model (Vitter, 1984), Triadic model (Yang, 1988), script-defined selective undo model (Prakash, 1994), direct selective undo model (Berlage, 1994), etc.

Event-object recovery model of Wang and Green is also a kind of nonlinear undo model (Wang, 1991). It divides the global recovery history into subhistories which corresponds to the objects of an object-oriented system. Our undo model discussed in this paper is derived from this model.

Although so many undo models have been proposed, it is the linear undo model that has been widely adopted in software products on the market. Let us regard the linear undo model adopted in those software products as traditional linear undo model so as to differentiate from our linear undo model. In the rest of this paper, we shall analyze the traditional linear undo model, point out its problems, and propose the solution.

2. TRADITINOAL LINEAR UNDO

2.1. Principle

In a traditional linear undo model, all the executed commands are recorded in a history list that may become arbitrary long (Figure 1). The last command in the list can be undone. When it is undone, it is removed from the history list and put into a redo list. The undo operation can be repeated until the history list empty (all of the commands are in the redo list). The last command put into redo list can be redone and again be appended to the history list. New commands are appended to the history list without affecting the redo list. It is important to note that undo and redo in this model are meta commands, they do not appear in the history (Berlage, 1994).

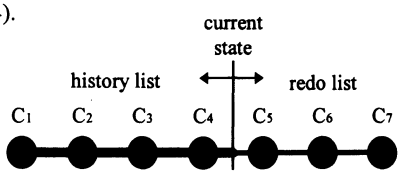

Figure 1. Traditional linear undo model

Sometimes linear undo model cannot work well. Suppose the user has drawn a circle on the screen and has later moved the circle to a new position. Now if the user undoes the move command and submits a new command to delete the circle, then the move command cannot be redone since the circle no longer exists.

The only case where linear undo model may not work well is when a new command is submitted while the redo list is not empty (Berlage, 1994). In some software products such as GINA and Microsoft Word, the redo list will be cleared when any new command is submitted. We adopt this method in our model as well. Another method to deal with this problem is to create new branches in recovery history (Figure 2).

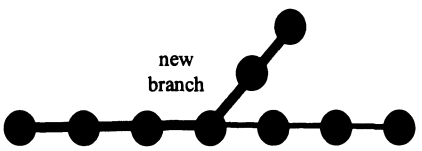

Figure 2. Creating a branch in the history

2.2. Advantages

Advantages of traditional linear undo model:

1. Its principle is simple and easy to be implemented.
2. Executing undo/redo by the order that the task-oriented commands have been submitted is in keeping with the imagination of users so that it is easy to be understood and accepted.
3. The user needn't know the details of commands. So its panel can be simplified as items of a menu. In contrast, for many nonlinear undo models, the commands stored in a recovery history must be displayed on screen so that the user can select a command from the history to undo or redo (Berlage, 1994 Myers, 1996). The nonlinear undo model may be hard to use for some end users who don't know much about commands.

2.3. Disadvantages

Disadvantages of traditional linear undo model:

1. The user can only execute undo/redo by the order that the task-oriented commands are recorded in the history list. However, the user may prefer to make his own choices during his recovery operations. Generally, the choices include,
   - to select an arbitrary command in the history list to undo/redo. For this purpose, many nonlinear undo models have been proposed. We don't discuss this problem in this paper.
   - to select an arbitrary object and change its states by undo/redo. Traditional linear undo model cannot support such operations. We mainly discuss this problem in this paper.
2. When a new command is appended to the history list while redo list is not empty, the linear undo model may not work well (Berlage, 1994). Two methods are available for this problem, to clear the redo list or create new branches in the history.
   - On clearing the redo list, some recovery information are lost so that further redo is impossible.
   - On creating new branches, the data structure of recovery history might become complex and unpredictable. The user has to make choice on a branch node. It would not be easy to use for some end users, so we don't adopt this method.

3. OBJECT-BASED LINEAR UNDO

As mentioned above, the user may prefer to select an arbitrary object and change its states by undo/redo. But the traditional linear undo model cannot support such operations. It is reasonable to reorganize the commands stored in a traditional linear history into subhistories based on the objects so as to support the user's purpose. To divide the traditional linear history into subhistories is not so simple as it is supposed. We must analyze the relationship among the commands in advance.

3.1. Relationship among commands

In order to analyze the relationship among the commands, we must know how the commands are submitted, executed and recorded.

The human-computer interaction can be described as a sequence of interactive cycles (Yang, 1988). Every interactive cycle has two phases, a planning phase and an execution phase. During a planning phase, the user evaluates the current state of an object, selects a command and inputs it. The submission of the selected command terminates the planning phase. The execution phase is the period during which the system executes the submitted command. The feedback from system terminates the execution phase.

Commands executed in the same interactive cycle are defined as synchronous commands. Commands executed in different interactive cycles but influencing each other are defined as asynchronous commands.

Generally, it can be supposed that only the command submitted by the user is executed in an interactive cycle. Therefore, it is reasonable to create a recovery history correspondingly to the sequence of interactive cycles by recording user-issued commands only. However, it should be noted that not only user-issued commands but also system-issued commands might affect the states of objects. Let us use the following example to explain why system-issued commands should be taken into account in creating an object-based recovery history.

Suppose there is a circular dependency (Vander Zanden, 1994) between rectangle A and B as follows.

A.right = B.left - 5
A.bottom = B.bottom
B.left = A.right + 5
B.bottom = A.bottom

In interactive cycle 1, the user may select A and move it to a new position. Then the constraint solver of the system evaluates the new values of B and moves B to a new position.

C_{a1} : Move(A, p_{a1}) *from user*
C_{b1} : Move(B, p_{b1}) *from system*

In interactive cycle 2, the user may select B and move it to a new position. Then the constraint solver of the system evaluates the new values of A and moves A to a new position.

C_{b2} : Move(B, p_{b2}) *from user*
C_{a2} : Move(A, p_{a2}) *from system*

In a traditional linear undo model, the history list of this example is $C_{a1}C_{b2}$. The constraint solver is also used to support undo and redo operations. When the user undoes C_{b2}, the state of A will be set back to its state before C_{a2} is executed automatically by the constraint solver of the system.

Now let us simply divide the traditional linear history $C_{a1}C_{b2}$ into subhistories based on objects A and B.

A: C_{a1}
B: C_{b2}

Suppose the user selects A to undo. In the subhistory of A, only C_{a1} is recorded so that only C_{a1} is to be undone. However, undo C_{a1} is meaningless because that the position of A has been changed by another command C_{a2} when C_{b2} was executed. The current position of A is not the result of C_{a1} but the result of C_{a2}. So C_{a2} should be also recorded in the subhistory of A in order to enable the user to select A to undo/redo. For the same reason, C_{b1} should be recorded in the subhistory of B.

A: $C_{a1}C_{a2}$
B: $C_{b1}C_{b2}$

Suppose the user selects A to undo. In the subhistory of A, C_{a2} can be undone. It is important to note that C_{a2} and C_{b2} are synchronous commands so that they should be undone or redone together in order to keep the dependency between A and B. So we need a method to record the synchronous relativity between C_{a2} and C_{b2}. Since synchronous commands are executed in the same interactive cycle, we can define a kind of record which can represent an interactive cycle. Now we are ready to build a new recovery model for object-based recovery operations.

3.2. Framework of our model

The framework of our model consists of three components (Figure 3). The Recovery Unit (RU) is the representation of an interactive cycle. All the objects manipulated in the same interactive cycle are recorded in the same recovery unit record. The Object Record (OR) describes the identity of an object created in an application. Each OR record corresponds to each object. The Recovery Information (RI) records the data necessary to execute recovery operation. A RI record corresponds to a RU record by the recovery unit identity code RUID.

The components are organized into three layers. The top layer is the list of RU records which represents the sequences of interactive cycles. The list of RU records is defined as main history which is similar to the recovery history of traditional linear undo model. The middle layer is the list of OR records. In the third layer there are different lists of RI records. One list of RI records belongs to one OR record uniquely and is defined as a subhistory. The linear undo mechanism is applied to both main history and subhistories. It is the subhistories that can support the user to select an arbitrary object to undo or redo.

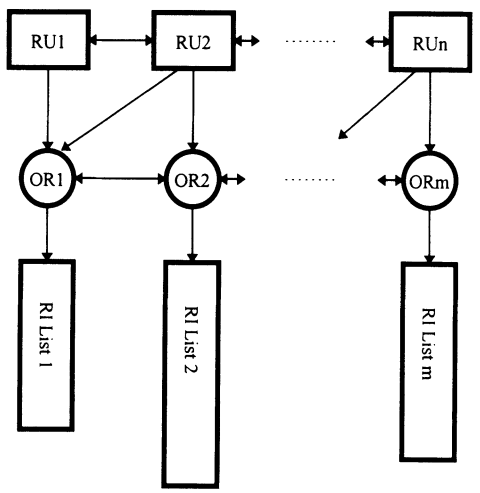

Figure 3. Framework of our model

3.3. Recovery operation

Our model allows the user to carry out undo/redo in both main history and subhistories. Undo/redo in main history is defined as global undo/redo. Undo/redo in subhistories is defined as local undo/redo. Let us use an example to explain how the framework of our model works. Suppose the user has operated as follows,

Step 1: CreateRect (A)
Step 2: CreateCircle (B)
Step 3: Move (A, p_{a1})
Step 4: Move (B, p_{b1})
Step 5: Select (A, B)
Step 6: Color ((A, B), red)
Step 7: Color ((A, B), green)
Step 8: Release (A, B)
Step 9: Color (A, pink)
Step 10: Color (B, yellow)

Where p_{a1} and p_{b1} are the positions on the screen. The commands working on A and B are divided into subhistories of A and B. The records of subhistories are illustrated in figure 4. The main history is shown in figure 5. We can see that object A and B are manipulated together from Step5 to Step8. The synchronous relativity between A and B from step5 to step8 is recorded in main history from RU5 to RU8. We define the area from RU5 to RU8 in the recovery history of our model as a synchronous zone (Figure 6).

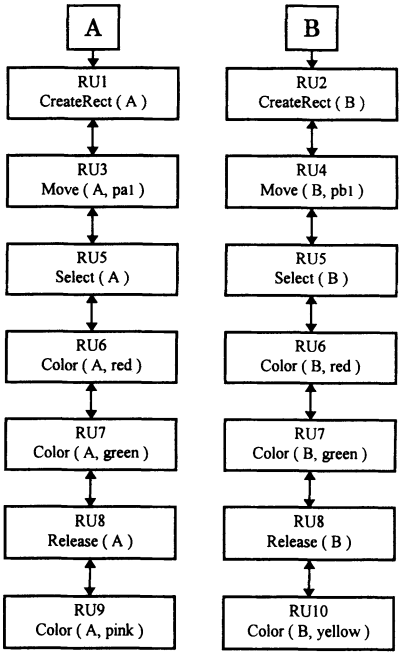

Figure 4. Records of subhistories

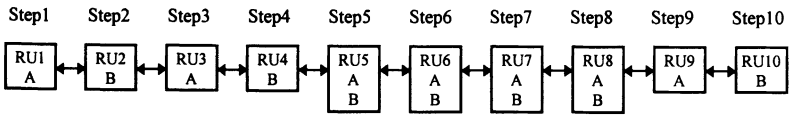

Figure 5. Records of main history

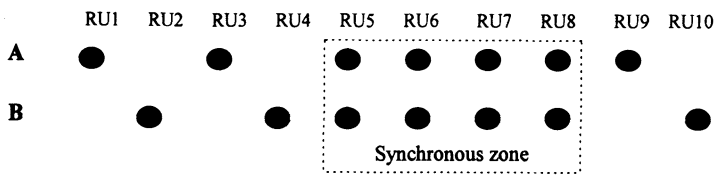

Figure 6. Synchronous zone in recovery history

Global linear undo/redo

The traditional linear undo mechanism (Figure 1) is applied to the main history of the object-based linear undo model. So the objet-based linear undo model can work as traditional linear undo model when the user do recovery operation in the main history only.

Suppose the user undoes the RU record identified as RU6 in the main history (Figure 5). The framework reads the set of objects recorded in this RU record. In this example, it reads object A at first and locates the subhistory of A to find the RI record identified as RU6 (Figure 4). If the command recorded in this RI record, Color (A, red), has not been undone, the framework undoes it, else does nothing. Then the framework reads the next object B and undoes it in the same way. This undo procedure continues until all the objects recorded in this RU record are undone. Naturally, redo function is implemented in the same way.

Global linear undo/redo method can preserve the dependencies among objects since it always undoes or redoes synchronous commands together.

Local linear undo/redo

Dependencies among objects should be considered when the traditional linear undo mechanism is applied to the subhistories of our model. For an independent object, its subhistory is independent. So the recovery operations can be executed in its subhistory independently without influencing each other with any other subhistory. For a dependent object, its subhistory relates to other subhistories. The undo/redo method of traditional linear undo should be modified to deal with this condition. The rules of local linear undo/redo is give as follows.

- when current undo/redo operation is out of any synchronous zone, the framework executes it in current subhistory independently.
- when current undo/redo operation goes into a synchronous zone, the framework displays a dialog box to remind the user that undo/redo operation is entering a synchronous zone and ask the user if he/she wants to undo/redo related objects together. On the answer 'Yes', it undoes or redoes other objects automatically. On the answer 'No', it undoes or redoes the selected object independently (Figure 8).

For instance, Objects A and B are independent from step1 to step4, dependent from step5 to step8, and independent again in step9 and step10 (Figure 5). It is obvious that A and B are independent out of the synchronous zone (Figure 6). So the states of A and B can be undone or redone independently out of the synchronous zone. A and B are dependent in the synchronous zone. Generally it is supposed that the dependencies among objects should be preserved. So the local linear undo/redo on A or B should be switch to global linear undo/redo on Both A and B in the synchronous zone.

Suppose both A and B are green and the user selects A to undo. At first, the framework locates the subhistory of A to find the current RI record which identified as RU7 and searches in the main history to find the RU record identified as RU7. Then the framework uses the global linear undo method introduced above to undo the commands Color(A, green) and Color(B, green) in the subhistories of A and B.

However, we have to note that the user may change his idea sometimes. For instance, when the user selects object A to undo/redo, the user may regard A as an independent object and undo/redo the states of A only. In other words, the user may want to break the dependency between A and B sometimes. Although it conflicts with the common assumption that the dependencies among objects should be preserved during recovery operations, we must support the user's purpose. Our solution to this problem is, when current operation of local linear undo/redo enters a synchronous zone, the framework asks the user if he/she wants to undo/redo the synchronous commands together. On the answer 'Yes', the framework will undo/redo synchronous commands automatically, else it will regards the current object as an independent object and undo/redo its states only. If the user changes his idea again and wants to keep the dependencies among objects, he can use the global linear undo/redo method or click on 'Yes' in the dialog box to make the framework switch local linear undo/redo to global linear undo/redo (Figure 8). Therefore the object-based linear undo model is flexible for the user to recombine the states of objects in a system.

Redo list

It is very often that the user does several steps of recovery operation and then submits a new command. For the traditional linear undo model, when a new command is submitted and appended to history list while redo list is not empty, one of the two methods would be taken: to clear the redo list or create a new branche in the history. We adopt the former method in our model.

When the new command is submitted out of any synchronous zone, only the redo list in current subhistory is cleared.

Suppose the user selects object A to undo (Figure 4). After several steps of undo operation, the current state of A is the state after the command 'Move (A, P_{a1})'. In the subhistory of A, the history list includes the RI records of RU1 and RU3, the redo list includes the RI records from RU5 to RU9. Now the user submits a new command 'Delete (A)'. This command is executed and its RI record is inserted at the position after the RI record of RU3, and then the redo list in this subhistory is cleared. In the main history (Figure 5), all the records of A from RU5 to RU9 are also cleared while the new RU record is inserted after RU3.

When the new command is submitted in a synchronous zone, the method on redo list depends on what kind of recovery operation that the user has chosen for this synchronous zone.

- If the user has chosen to undo/redo the current object only, then the new command can affect on the current object only because the synchronous dependency has been broken. If the redo list in the subhistory of current object is not empty, the framework will clear it and then clear its corresponding records in the main history.
- If the user has chosen to undo/redo synchronous commands together in this synchronous zone, the new command can affect on all the objects in this synchronous zone. The framework will assign the new command into the subhistories correspond to the objects. If any redo list in these subhistories is not empty, the framework will clear it and then clear its corresponding records in main history.

4. IMPLEMENTATION

The implementation of our model is based on TRIBASE, a UIMS system which is developed by SEIKO Instrument Incorporation (SEIKO, 1995). TRIBASE can support cooperation among multiple applications. It has been applied to commercial software products such as VLSI CAD systems.

The implemented system consists of three parts (Figure 7). The kernel part is TRIBASE which manages interfaces of applications through the user interface database (UIDB) and supports communication among applications. The sample application (SP) is a simple graphical editor which has its own UIDB records and a Command Reference Table that contains the definition of primitive commands and corresponding reverse commands. The recovery handler (RH) has its own UIDB records as well. The framework of our model is built in RH.

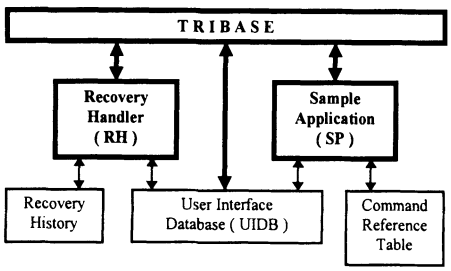

Figure 7. Architecture of the implemented system

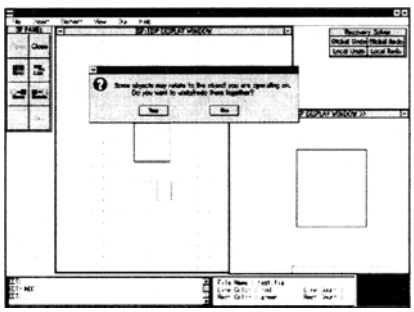

Figure 8. An example display of the sample application with the recovery handler panel and the synchronous undo/redo confirmation dialog box.

When SP begins to run, its application ID and Command Reference Table are sent to RH. When the user works on SP, the primitive commands are packed into recovery messages and sent to RH. All the recovery messages are written in a standard format. When a recovery message is transmitted from SP to RH, Do procedure of RH is invoked to create corresponding records of recovery history. When the user works on RH with undo or redo commands, the reverse commands or original commands are packed into recovery messages and sent to SP to be executed without creating any recovery message. Figure 8 is a sample display.

5. RELATED WORK

Our model is derived from the event-object recovery model of Wang and Green (Wang, 1991). In their model, the global history is divided into subhistories and encapsulated in corresponding objects. The framework is composed of a set of Recovery Objects (RO) which are nested in the user interface objets and organized in a predefined hierarchical structure. Each RO has its own list of Recovery Information (RI) and accepts recovery commands. The undo/redo method for each RI list is similar to US&R model. We use the concept of recovery unit of this model to represent interactive cycles and deal with synchronous commands. However, instead of encapsulating recovery histories into corresponding objects, we organize main history and subhistories into a global control framework. In this way, we can separate recovery utility from concrete applications. The separation makes it possible for us to design a common recovery handler which can serve multiple applications.

6. CONCLUSION AND FUTURE WORK

The object-based linear undo model proposed in this paper is derived form both traditional linear undo model and event-object recovery model. The framework of this model consists of a main history which represents the sequence of interactive cycles and subhistories which record the operations on corresponding objects. The main history can support traditional linear undo/redo operations and the subhistories can support object-based recovery operations. The framework of our model is flexible for the user to recombine the states of objects. Furthermore, the framework of our model can be separated from concrete applications. The separation makes it possible to design a common recovery handler which can serve multiple applications. It should be mentioned that the framework of our model can support not only linear undo mechanism but also nonlinear undo mechanism. We plan to propose object-based nonlinear undo model and improve object-based linear and nonlinear undo models to support collaborative systems.

7. REFERENCES

Archer, Jr., J. E., Conway, R. and Schneider, F. B. (1984), *User Recovery and Reversal in Interactive Systems*, ACM Trans. on Programming and Systems, Vol.6, No.1, January 1984, 1-19.

Berlage, T. (1994), *A Selective Undo Mechanism for Graphical User Interface Based on Command Objects*, ACM Trans. on Computer-Human Interaction, Vol.1, No.3, September 1994, 269-294.

Myers, B. A. and Kosbie, D. S. (1996), *Reusable Hierarchical Command Objects*, CHI 96, April 13-18, 1996, 261-267.

Prakash, A. and Knister, M. J. (1994), *A Framework for Undoing Actions in Collaborative Systems*, ACM Trans. on Computer-Human Interaction, Vol.1, No.4, December 1994, 295-330.

SEIKO Instruments Incorporation (1995), *TRIBASE Manuals*.

Vander Zanden, B., Myers, B. A., Giuse, D. A. and Szekely, P. (1994), *Integrating Pointer Variables into One-way Constraint Models*, ACM Trans. on Computer-Human Interaction, Vol.1, No.2, June 1994, 161-213.

Vitter, J. S. (1984), *US&R: A new Framework for Redoing*, IEEE Software, Vol.1, No.4, October 1984, 39-52.

Wang, H. and Green, M. (1991), *An Event-object Recovery Model for Object-oriented User Interfaces*, UIST 91, November 11-13, 1991, 107-115.

Yang, Y. (1988), *Undo Support Models*, Int. J. Man-Machine Studies **28**, 457-481.