A Regional Undo Mechanism for Text Editing

Rui Li and Du Li
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112 USA

ABSTRACT

As a powerful error recovery facility, selective undo has been proposed by many researchers to undo arbitrary operations in interactive applications. However, existing approaches in the context of text editing generally fail to address adequately the problem of how the user can select the operations to undo. We describe a novel regional undo technique to address this problem, which allows the user to dynamically select regions from some concrete document state. The operations to undo, that fall partly or completely into the regions, are identified by the system automatically. Regional undo effectively overcomes the limitation of traditional selective undo approaches which require the user to manually pick up operations to undo from the operation history.

KEYWORDS: Regional Undo, Selective Undo, Error Recovery, Text Editing, Operational Transformation

INTRODUCTION

Undo is a useful and widely-supported feature in single and multiuser interactive applications. As a backward error recovery mechanism, an undo facility enables explorative learning without fear, which is important in a world where new and improved software features and tools appear frequently [1] [2]. Undo is especially important for text editing, because it is so easy to make mistakes in an editing process. In fact, a single mouse click or typo is sufficient to cause undesirable document changes and the user may not always be able to immediately realize the mistakes they have made.

Two undo mechanisms have been proposed, namely, non-selective undo and selective undo. Non-selective undo, also called linear undo, allows the user to undo executed operations in inverse chronological order. The user can repeatedly undo the last operation in the operation history until the history becomes empty. Linear undo is widely implemented in most text editors such as MS Word and Vi. However, linear undo may cause the loss of work if the operations to undo are not exactly the last in history. Therefore, selective or non-linear undo is necessary so that the user is allowed to only undo the operations that caused the mistakes [2] [8].

In selective undo, it is critical for the user to know how to select the operations to undo from the operation history. Unfortunately, little attention has been paid to this aspect in the text editing literature. Existing research (e.g., [8] [10]) has developed numerous algorithms to ensure the correct execution of selective undo once the operations to undo are selected. However, it remains *the problem* how the users can conveniently locate the desired operations to undo.

A direct way to address this problem is to visualize the operation history so that it is hopefully easier for the user to select operations to undo. Existing approaches can be divided into two categories, namely, textual and graphical history. In the textual history approach, e.g., as in GINA [2] and Amulet [7], the user may pick up operations to undo by browsing and searching on the operation history. Unfortunately, it is generally difficult for the user to recognize the exact names of operations and reconstruct document states from the textual representation of operations that are performed through direct manipulation [6]. For editing operations with position parameters, the positions are defined on the document states on which they were performed. As the document state changes over time, the position values may not make sense to the user at all. Nonetheless, this is the approach generally assumed in existing textual editors that support selective undo, e.g., DistEdit [8] and REDUCE [10].

The second approach is to visualize the operation history as the "snapshots" of the graphical results of executed operations, as in Kurlander and Feiner [5] and Meng et al. [6]. The user browses on these snapshots to search for operations to undo. This approach avoids the problems associated with the cryptic textual representation of operations. Meng et al. [6] further provides a powerful history searching and grouping facility so that the user can choose to focus on the part of the operation history that is related to selected objects and then perform undo on the partial history. This approach works well in editors where objects in the document are cleanly identifiable. However, its applicability is rather doubtful in text editors where the document is a stream of characters and the concept of object is not well-defined and heavily subject to the interpretation of specific applications.

Nevertheless, visualizing the change history (snapshots) of a textual document as caused by fine-grained operations may confuse rather than help the user at all, especially when the document is large. Moreover, the contents a user intends to recover may not always map to particular operations directly. For example, the user may only want to keep part of a paragraph that was pasted or to recover part of paragraph that was cut by mistake. That is to say, it is often necessary to undo part of an operation. Unfortunately, previous selective undo approaches in general treat performed operations as atomic units that are not further divided in the undo/redo process.

In this paper, we contribute a novel regional undo technique to address the above problems. Regional undo, as one kind of selective undo mechanism, allows the user to perform undo on arbitrarily selected regions in some concrete document state. The system automatically selects operations that affect the designated regions, without requiring the user to manually select operations from the operation history. The advantages of regional undo are obvious: First, focusing on regions can incur much less cognitive overhead than performing undo on the entire document. Second, selecting regions from a given document is easier for the user than selecting the cryptic operations from the history. In modern document authoring tools, regions can often be augmented with mnemonic audio and visual cues, as explored in the Data Mountain project [9]. While selection of regions can be incrementally refined, selection of operations must be accurate.

To our best knowledge, Prakash and Knister [8] is the first (and the last) work in the text editing literature to introduce the concept of regional undo. However, no concrete discussion or technical detail is given in [8]. In fact, their work only allows the user to select regions in current document state and the original operations are considered as atomic units. In contrast, our approach allows the user to select regions in any intermediate document state and partially undo a previously executed operation. Besides, the range of a region can be incrementally refined in our approach when the user does not remember the exact undo region in the beginning.

The rest of this paper is organized as follows. We first motivate regional undo through a concrete text editing example in the second section. The third section introduces regional undo from the user's point of view. Then the following two sections present the technical details of regional undo. A complete example is given in the sixth section to illustrate the whole regional undo process. The seventh section develops an interpretation of regional undo in the broader context of error recovery. This is followed by a comparison to major related previous works. We then conclude this paper and point out future research directions.

A MOTIVATING EXAMPLE

We begin by presenting a simple example to motivate why we need regional undo. Consider a scenario when the user tries to input the following sentences:

"David Kurlander and Steven Feiner. A history-based macro by example system. In *User Interface Software and Technology*, 1992."

After the user inserts these three sentence, the words "macro by example" are deleted due to a wrong operation. However, the user does not realize this mistake immediately and continue to input other words. After a while, the user finally realizes the problem and wishes to rectify it by undo. However, the available linear undo mechanism does not help at all because the user has written some other contents and does not want to lose them. The document state is now as follows:

"...David Kurlander and Steven Feiner. A history-based system. In *User Interface Software and Technology*, 1992. ..."

The example actually represents a class of cases that often arise in text editing, in which the user knows exactly what operations should be undone. In the above example, the user knows exactly that he can reach his goal by undoing the operation which deletes "macro by example". If the user could realize the errors earlier, he would be able to easily correct them by picking up and undoing the erroneous operations from the operation history. However, the user is often not able to realize the mistakes until after working on the document for a long time. Then browsing the operation history to identify the erroneous operations (backward error recovery) becomes more time-consuming than directly editing on the erroneous contents (forward error recovery). As a result, the user is often not motivated to use selective undo for error recovery [1].

Moreover, the user may encounter more complicated situations where the mistakes cannot be easily rectified by forward error recovery. For example, when people try to address a difficult issue, the thought process is not always straightforward. During this process, certain replaced ideas may be recognized later as being more appropriate than current ideas. Then the user may hope to recover the original descriptions of the replaced ideas rather than to rewrite them because he cannot remember how they are described before. Such a case is different from the previous case by nature. Now the user often has to face new difficulties: He cannot exactly remember the contents to be recovered, or he cannot remember what operations have affected these contents to be recovered. In other words, the user does not know what operations should be selectively undone to achieve his intention. However, exiting selective undo mechanisms do not seem to help at all.

In fact, the above problems are aggravated in multi-user collaborative editing, where the users may be allowed to work on different parts of the same document at the same time. A user may not be able to realize the changes made by collaborators on time. Since such content changes come from other users, it is very hard, or even impossible, for the user to know exactly which operations are responsible for the undesired changes, not to mention to selectively undo them.

REGIONAL UNDO OVERVIEW

The examples in last section actually illustrate the limitations of existing solutions of selective undo. The root of the problem is that existing approaches are operation-oriented, which introduces two difficulties for the user. Firstly, to recover the earlier errors, the user are required to at least know what operations lead to the errors. This requirement is generally too demanding for the user, due to a significant gap between the cryptic representation of operations and the document state that the user can intuitively reconstruct. Secondly, even though the user knows the operations which lead to the errors, it does not mean that the undo task becomes easier, because the user has to manually find and undo them in the operation history. All performed operations are normally stored in the operation history in chronological order. But, the desired operations to undo are not always chronologically successive in many situations. If there is no intuitive help from the system, selecting the operations to undo becomes a painful process.

The idea of regional undo is developed in this paper to overcome the drawbacks of existing approaches to selective undo. Regional undo is content-oriented rather than operation-oriented. The user does not need to understand the operation history any more, if the purpose is to recover earlier errors. Once the user selects a proper region, the system will automatically select only the the operations that overlap this region and undo them according to the user's requirement. Compared to directly selecting operations in the operation history, selecting region is more user-friendly and incurs less cognitive overhead, as analyzed in the introduction.

To better explain regional undo, we first give some related definitions. When processing regional undo, the region the user selects in the current document state is called the *initial undo region*, whereas the region the user wants to recover from the previous state just prior to the erroneous operation is called the *goal undo region*. In the above example, the region that "A history-based system" occupies is the initial undo region based on the current document state. And the region that "A history-based macro by example system" occupies based on the document state just before the unexpected delete operation is the goal undo region.

According to whether the proper initial undo region is known or not, we divide regional undo into two categories: regional undo with known region and regional undo with unknown region. Here, the proper initial undo region means that the effect of processing undo in this region selected in the current document state just recovers the desired contents without loss of other contents. Back to the above example, when the user finds our that he made a mistake, the user may select the region where "A history-based system" is and process regional undo to remove the delete operation performed in this region. So, in this case, regional undo with known region can recover the user's error.

The user employs regional undo with known region to seek his goal. The user is required to first select an initial undo region. After that, the user starts the undo procedure by clicking the undo button. Every time the user clicks the undo button, the system automatically looks for the next available operation. If such a operation does not exist, the undo procedure stops, regardless of the number of clicks on the undo button. Otherwise, the found operation is undone. All subsequent operations are redone in the document state just before this effective operation is performed. In the meantime, the undo region is updated.

Since the precondition that regional undo with known region based on is that the user can find the proper initial undo region. Unfortunately, this is not always a reasonable assumption. In the second case discussed in last section, this approach cannot work well, because the user is not clear about the proper initial undo region. For example, the user pastes the following texts in the current document state.

"A history-based macro by example system."

And after typing "David Kurlander", the user deletes this sentence, because this user might think that this sentence is not necessary. Later, after the user realizes that this deletion is a wrong decision, he has typed other contents, "and Steven Feiner." Regional undo with known region does not work in this case. The reason is that the user cannot select the proper initial undo region in the current document state so that undoing in this region only recovers the desired sentence without losing other contents.

To solve the problem, we introduce regional undo with unknown region. Unlike regional undo with known region, the knowledge of the proper initial undo region is not necessary. Since the user is not sure about the initial undo region, he is allowed to set the goal undo region in some previous document state and then process regional undo in this selected region. Once the user is able to figure out the content to be recovered, regional undo can always succeed.

Since the contents to be recovered is based on a previous document state, searching for them is not an easy job. Here we introduce the concept of temporary undo to achieve the goal. Temporary undo means that the user's undo actions only happen in the application view, not in the internal data structure. Hence when processing a temporary undo, the modifications caused by undo only happen to a copy of the operation history. Only when the user commits the actions do the modifications caused by temporary undo affect the internal data structure. Since there is no modification in the internal data structure, the user can browse the previous document state by temporary undo without fear. Temporary undo actually provides a kind of mechanism by which the user may conveniently look for the contents to be recovered in the previous document states.

After selecting the goal undo region in a previous document state, the user can process regional undo in this region by simply clicking the regional undo button. The user's actions result in that only the operations that fall related to the undo region are really undone and other operations will be redone. Fortunately, the actual undo/redo work is automatically done by the system in background and the user is free from care.

Look at the previous example again. Using regional undo with unknown region, the user will first perform temporary undo until the contents to be recovered, "A history-based macro by example system", come back to the document context. Then the user selects the region covering exactly the sentence as the goal undo region and performs regional undo. After that, other operations not affecting the goal undo region will be automatically redone.

Note that if the selected initial undo region in regional undo with known region or the selected goal undo region in regional undo with unknown region is empty, regional undo does not cause any change to the document context. The reason will be explained in the next section.

REGION TRANSFORMATION

In the following two sections, we present the technical details of regional undo. First in this section we introduce some basic concepts necessary for the reader to understand how operations relative to a given region are singled out from the operation history. Then in the next section we discuss how these operations are undo, completely or partially.

Region Definition

As in well-known text editors such as Grove [4], DistEdit [8], and REDUCE [10], we model the textual document in question as a linear string. If there are n characters in the document, we define that

(1) the position before the first character is 0;
(2) the position before the i-th character is i-1; and
(3) the position after the last character is n.

If a region R is between positions a and b, inclusively, it is denoted as $R[a, b]$, where a must be smaller than or equal to b. If a equals b, the region is empty. The length of region R is denoted as L(R) = $b - a$. In addition, we use R.begin to denote the begin position of region R and R.end to denote its end position, which are a and b, respectively.

Relationship between Regions

We consider the following six cases when comparing two regions, R1[a,b] and R2[c,d], given they are defined on the same document state, as shown in Figure 1.

(1) R1 precedes R2, only if $b \leq c$.
(2) R1 front-overlaps R2, only if $a \leq c < b$ and $b < d$.
(3) R1 is contained by R2, only if $a > c$ and $b < d$.
(4) R1 rear-overlaps R2, only if $c < a < d$ and $b \geq d$.
(5) R1 follows R2, only if $a \geq d$.
(6) R1 contains R2, only if $a \leq c$ and $b \geq d$.

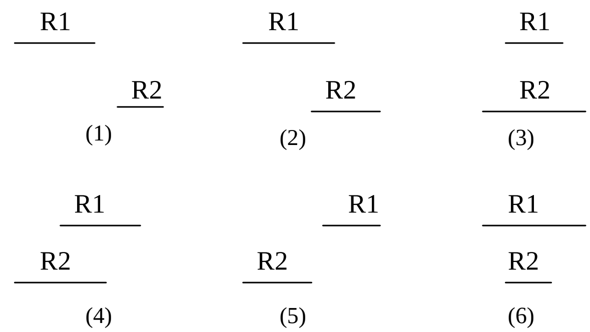

Figure 1: Six region relations between two regions, R1 and R2: (1) R1 precedes R2. (2) R1 front-overlaps R2. (3) R1 is contained by R2. (4) R1 rear-overlaps R2. (5) R1 follows R2. (6) R1 contains R2.

Relation between Operations and Regions

According to the above region definition, the length of a region is determined by the begin and end positions of this region. Editing operations will necessarily affect the begin and/or end positions of a selected region. For example, if an insert (or delete) operation is performed before the region, then both the begin and the end positions of this region should be increased (or decreased). On the other hand, if the operation is performed after the end position of this region, the region will not be affected.

Due to the relation between operations and regions, we can always trace a region given in one document state to any other document states. Such a procedure is defined as *region transformation*. We can also identify in the operation history those operations that completely or partly fall in a given region and affect the content of this region. Those operations are called *effective operations*. Region transformation is the basis of regional undo. When the user performs undo on a given region, the system automatically finds all the effective operations by region transformation.

Given a region defined in one document state, we often need to calculate the form (or state) of the region in another document state. We call the former *the initial state* and the latter *the goal state*. The region position parameters need to be transformed against the sequence of operations which are performed between these two document states. If the initial state is earlier than the goal state, the region is transformed against those operations in chronological order. Otherwise the region is transformed against the operations in inverse chronological order. In processing transformation of a region against an operation, we need to compare their position parameters.

Operation Region

We consider the following three classes of operations in text editing. The first class is called text-increasing operations, such as the insert operation. The second class is the text-decreasing operations, such as the delete operation. The third class is called text-constant operations, such as those for formatting and highlighting. For text-increasing operations, the length of an operation is the number of inserted characters. For text-decreasing operations, the length of operations is the number of characters to be deleted. For a text-constant operation, its length is the number of characters which this operation affects. Here we use L(O) to denote the length of an operation O. In addition, the begin position of O is denoted as P(O). We define the concept of operation region OR(O) of a given operation O as follows:

(1) Text-increasing operations: In inverse chronological region transformation, OR is defined as [a,b], where $a = P(O)$ and $b = P(O) + L(O)$. In chronological region transformation, OR is defined as [a,b], where $a = b = P(O)$.

(2) Text-decreasing operations: In inverse chronological region transformation, OR is defined as [a,b], where $a = b = P(O)$. In chronological region transformation, OR is defined as [a,b], where $a = P(O)$ and $b = P(O) + L(O)$.

(3) Text-constant operations: No matter whether region transformation is chronological or inverse chronological, OR is always defined as [a,b], where $a = P(O)$ and $b = P(O) + L(O)$.

In addition, we should consider the replace and paste operations which are widely implemented in text editors. Similar to insert and delete operations, these two operations also change the number of the characters in the text document. In our model, they are simply translated into delete and insert operations.

Region Transformation in Chronological Order

Given a region R[a,b] defined in a document state, in order to trace its content in a later document state, R is transformed against all operations between these two document states in chronological order. Consider one such operation O, whose begin position is P(O) and length is L(O).

(1) Text-increasing operation: If the operation region precedes region R, the begin and end parameters of R is increased, i.e., $R.begin = a + L$ and $R.end = b + L$. If the operation region is contained by R, the end position of R is increased, i.e., $R.end = b + L$. If the operation region follows R, then R does not change. For example, the initial document state is "abcdefg", the selected region covers "cde". And an insert operation is performed to insert "xy" after d. The document state becomes "abcdxyefg". The region relative to the current document state now covers "cdxye". Note that in this case the operation region will not front-overlap, rear-overlap, or contain the given region.

(2) Text-decreasing operation: The situation is slightly complicated. First we denote the overlapping region between R and the operation region as VR. If the operation region precedes R, we have $R.begin = a - L$ and $R.end = b - L$. If the operation region front-overlaps R and $VR = [a, x]$, then VR is deleted and $R.begin = P$ and $R.end = P + (L(R) - L(VR))$. If the operation region is contained in R, then VR is deleted and $R.end = b - L(VR)$. If the operation region rear-overlaps R, then VR is deleted and $R.end = b - L(VR)$. If the operation region follows R, then R is not changed. If the operation region contains R, then $R.begin = R.end = P$.

For example consider that the initial document state is "abcdefg" and the selected region covers "cde". After a delete operation is performed to delete "abcd", the current document state is "efg". The region relative to the current document state only covers "e".

(3) Text-constant operation: The region does not change.

Region Transformation in Inverse Chronological Order

When a region is transformed against a text-increasing operation, it is the same as the text-decreasing case of region transformation in chronological order. If the operation is a text-decreasing operation, it will be the same as the text-increasing case of region transformation in chronological order. If the transformed operation is a text-constant operation, the region does not change.

Empty Region

We consider the case when a selected region becomes empty due to delete operations. For example suppose the initial document state is "abcd". After the user inserts "xy" after 'c', the document state becomes "abcxyd". Assume that the user selects the region covering "xy" in the current document state. This region is from position 3 to 5. Now the user first deletes "xy" and then inserts "mn" after 'b'. The document state becomes "abmncd". However, the selected region becomes empty. After the content "xy" is deleted, the begin and end positions of the region are both 3. After inserting "mn", the region does not disappear although it remains empty. The begin and end positions of this region now become 5.

However, the empty region possibly incurs ambiguity. Back to the above example, what will happen if the user inserts "mn" not after 'b' but 'c'? In fact, when the empty region is transformed against the operation of inserting "mn", there exist two possible results. The empty region is either between 'c' and 'm' or between 'n' and 'd'. In other words, if "xy" is recovered, "xy" could be before "mn" or after "mn". Since the user inserts "mn" after "xy" has been deleted, both these two results are acceptable. To remove the ambiguity, we consistently define that "xy" appears before "mn" once it is recovered.

Filtering Effective Operations

In regional undo, region transformation is used as a filter to select effective operations. Let's use an example to explain how it works (Figure 2). The initial document state is empty. After executing five operations, the current document state becomes "zaxcd". Assume that there exists a region covering "ab" in the state just prior to executing O_2. Which are the

Performed Operations	**Document State**
O1 = insert “ab”	“[ab]”
O2 = insert “cd” after “b”.	“[ab]cd”
O3 = insert “xy” before “b”	“[axyb]cd”
O4=delete “yb”	“[ax]cd”
O5 = insert “z” before “a”	“z[ax]cd”

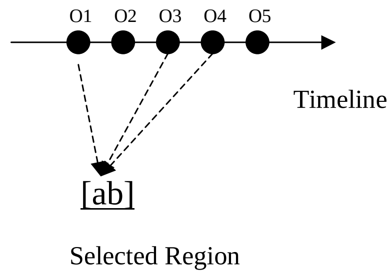

Figure 2: An example of operations filtering.

effective operations of this region from O_2 to O_5? We use region transformation to find them. First, the selected region is compared with O_2. Since O_2 does not overlap the selected region, it is not an effective operation. After processing region transformation against O_2, the selected region does not change. Next, the selected region is compared with O_3, since O_3 happens in the middle of the selected region, O_3 is an effective operation. Processing region transformation against O_3 will extend the selected region to cover “axyb”. Similarly we can conclude that O_4 is also an effective operation while O_5 is not.

Using region transformation, the system can detect all the effective operations relative to a given region, although they are possibly not successive in time. In the above example, the system identifies only three effective operations, O_1, O_3 and O_4, from the operation history. Then only these three operations are involved when the user performs undo/redo on the selected region “ab”.

Note that if the initially selected region is empty, the effective operations of this region do not exist. This is because that an empty region cannot be transformed into a non-empty region. Hence if the user processes undo on an empty region, nothing will happen.

REGION OVERLAPPING AND PARTIAL UNDO

In regional undo, the user usually cannot ensure that the region selected completely covers the operation regions of its effective operations. We define this problem as region overlapping. Region overlapping cannot be avoided. For example, assume that a user first pastes a string “abcdef” in an empty document state. And then this user selects a region from ‘b’ to ‘d’. In this case, the region only covers part of the paste operation. Therefore it is an important issue in regional undo how to deal with this situation.

Partial Undo

In this paper, we adopt the the following policy to handle this problem, to better satisfy the user’s intention.

If an operation partly falls in the given undo region, only overlapped part of this operation is undone.

Based on this policy, we introduce *partial undo*, which means that part of an operation can be involved in undo action. For example, if a user pastes three sentences in one operation, this user only wants to keep the first sentence a while later. In regional undo, the user first selects the region where the second and third sentences are and processes undo in this region to remove the last two sentences. Of course, the user may use delete operation to achieve the same result if the regional undo mechanism is not available [1].

Performed Operations	**Document States**
O1 = insert “abcd”	“abcd”
O2 = insert “efgh”	“[abcd]efgh”
O3 = delete “[abcd]efgh”	“”

Overlapping Region

Figure 3: A partial undo example in regional undo with unknown region.

In fact, partial undo is also very useful in regional undo with unknown region. Look at another example (Figure 3). The current document state is empty. Assume that the user wants to recover the deleted content “abcd” by undo. However, if the user only undoes O_3, while “abcd” is successfully recovered, “efgh” is also recovered although they are not what the user expects. In fact, it is generally difficult for the user to remember the effect of every operation. If the effect of the involved delete operation includes the contents to be recovered as well as other contents, undoing this operation itself is not enough. Partial undo can be used to solve the problem. In this example, after the user is back to the previous state “abcdefgh” and selects the goal undo region just covering “abcd”, the user processes regional undo in this region. According to our policy, only the overlapping part of O_3 with the undo region is undone. As a result, only “abcd” is recovered. The user’s intention is achieved.

Splitting and Combining Operations

Since the users are allowed to undo part of an operation, operations, sometimes, need to be dynamically split to be several parts. Of course, only a part in the given undo region is undone. The newly generated parts replace an original operation to be stored in the operation history. Assume that there exists the operation history (OH):

$$OH = [O_1, ..., O_i, O_{i+1}, ..., O_n].$$

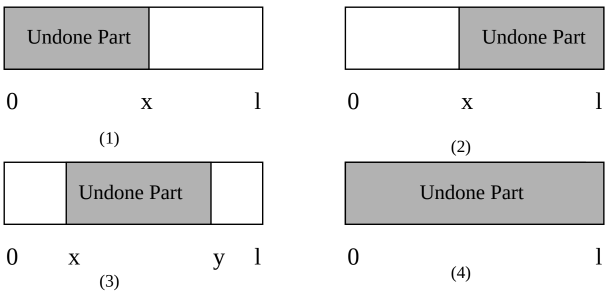

Figure 4: Four situations in undoing an operation. The whole part represents the operation region of an involved operation and black part represents the undone part of this operation.

If O_i is undone, there exist four situations, see Figure 4. Assume that the number of the characters that O_i affects is l.

(1) The undo region front-overlaps the operation region of O_i. Assume that this part includes the first x characters, where $x > 0$ and $x < l$. In this situation, O_i is split to two same types of operations, O_i^1 and O_i^2. The characters that O_i^1 affects are the first x characters. The characters that O_i^2 affects are from $x+1$ to l. Only O_i^1 is undone.

(2) The undo region is contained by the operation region of O_i. Assume that the this part includes characters from $x+1$ to y, where $x > 0$ and $y < l$. In this situation, O_i is split to three same types of operations, O_i^1, O_i^2 and O_i^3. The characters that these three operations affect are respectively from 1 to x, from $x+1$ to y and from $y+1$ to l. Only O_i^2 is undone.

(3) The undo region rear-overlaps the operation region of O_i. Assume that this part includes the characters from $x+1$ to l, where $x > 0$ and $x < l$. In this situation, O_i is spilt to two same types of operations, O_i^1 and O_i^2. The characters that they affect are respectively from 1 to x and from $x+1$ to l. Only O_i^2 is undone.

(4)The undo region contains the operation region of O_i. O_i is undone.

Except the fourth case, the first three cases mean that only part of the operation is undone. Without loss of generality, we only consider most complicated case, the second case, in which O_i is separated to three parts, denoted as O_i^1, O_i^2, O_i^3. These operations are of the same types. And they are contextually serialized. It means that the document state O_i^2 based on is the state just after O_i^1 is performed. And the document state O_i^3 based on is the state just after O_i^2 is performed. In the operation history, We append O_i^1, O_i^2 and O_i^3 adjacently after O_i. Meantime, O_i is marked as a skipped operation, denoted as O_i^*. Such marked operations are ignored in subsequent transformations.

As an example, assume that the initial document state is "abcd". And a user pastes a string "xyz" after "b". The document state becomes "abxyzcd". Now, this user performs regional undo in the given region just covering "xy". Since the undo region covers part of the pasted contents, according to our policy, only "xy" are undone. In performing regional undo, the original operation, O = insert("xyz",2) is first replaced by two successive operations, O^1 = insert("xy",2) and O^2 = insert("z",4). And then only the operation O^1 is undone. Now, we know which operation should be undone. We will continue to discuss how to undo these operations in following subsection.

Undoing Selected Operations

In earlier sections, we focus on how to select the desired operations by region transformation to implement the users' intention. In this subsection, we discuss how to undo the desired operation in detail. Our approach is based on operational transformation [4].

We adopt the script-based interpretation of undo [2]. That is, the effect of undoing O is equivalent to skipping O and redoing all subsequent operations. When undoing an operation O, its inverse operation $\overline{O}$ is generated and transposed to the end of the operation history to get $\overline{O}'$. Then $\overline{O}'$ is generated and executed in the current document state. The operation history is modified so that the inverse operation of O is stored adjacently after O. Due to this change, all subsequent operations in the operation history need to be transformed against $\overline{O}$. In the meantime, O and $\overline{O}$ together are considered as a do-undo pair. Do-undo pairs in the operation history are equivalent to ignoring O. Other operations do not need to be transformed against them.

For example, if the initial document state is empty and the user inserts three successive operations, O_1 = insert("ab",0), O_2 = delete("a",0) and O_3 = insert("cd",1). The effect of these operations changes the document state to be "bcd". Now the user wants to recover a by undoing delete operation, O_2. Thus, the inverse operations of O_2, $\overline{O_2}$ = insert("a",0), is generated. And it is transformed against O_3 to get a new operation, $\overline{O_2}'$ = insert("a",0). After $\overline{O_2}'$ is performed on the current document state, a is recovered. Meantime, O_3 is transformed against $\overline{O_2}$ to get O_3' = insert("cd",2). The operation history is updated as:

$$OH = [O_1, O_2, \overline{O_2}, O_3'].$$

Of course, if partial undo is involved, the situation is more complicated. Let's continue with the example discussed in

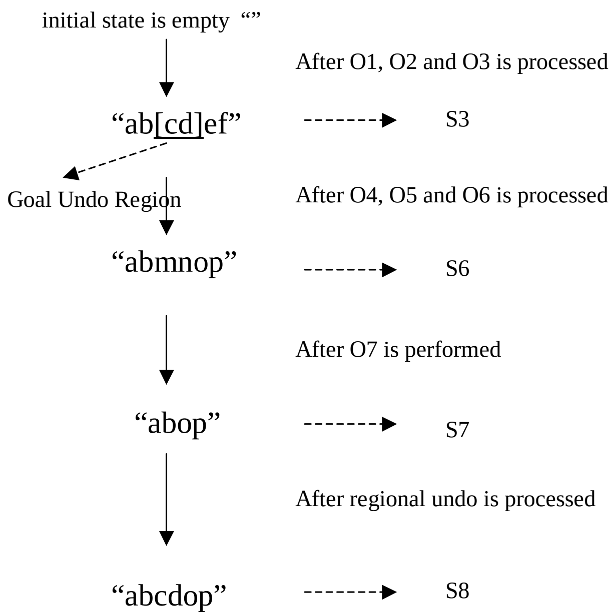

Figure 5: A Complete Example

previous subsection. Since O is split into O^1 and O^2, it is marked in the operation history. To undo O^1, the inverse operation of O^1 is generated, denoted $\overline{O^1}$ = delete(“xy”,2). This operations is transposed against O^2. $\overline{O^1}$ and O^2 are respectively updated to be $\overline{O^1}'$ = delete(“xy”,2) and $O^{2'}$ = insert(“z”,2). After $\overline{O^1}'$ is performed in the current document state, the document state becomes “abzcd”. The user’s intention is satisfied. Meantime, the operation history is updated. The inverse operation of O^1, $\overline{O^1}$, is also stored in the operation history.

$$OH = [..., O^*, O^1, \overline{O^1}, O^{2'}].$$

In addition, when the user processes redo, the corresponding inverse operation will be removed and operation combing procedure is invoked. All adjacent non-inverse operations need to be combined to be a new operation. After that, these non-inverse operations will be removed from the operation history. If all split operations have been combined to be the original one, these split operations will be removed from the operation history and the star mark of original operation will be removed.

A COMPLETE EXAMPLE

In this section, we use an complete example to explain how regional undo works. Since regional undo with unknown region is more complicated than regional undo with known region, we only give an example related to regional undo with unknown region (see Figure 5).

The initial document state is empty. The user does a sequence of operations described below. To conveniently present, all involved operation is string-based. At first, the user performs three operations, O_1, O_2 and O_3 in sequence. They are respectively to insert(“ab”,0), insert(“cd”,2) and insert(“ef”,4). After that, the document state is S_3 in Figure 5.

Then, the user performs one delete operation O_4 = delete(“cdef”,2), effect of which is to delete “cdef”. After that, the user performs two insert operations O_5 = insert(“mn”,2) and O_6 = insert(“op”,4) again. Now, the document state is S_6 in Figure 5.

After these works are done, the user deletes “mn” (O_7 = delete(“mn”,2)) and expect to recover “cd” inserted before. At that moment, the operation history includes seven operations as following.

$$OH = [O_1, O_2, ..., O_7]$$

The user cannot select the proper initial undo region in the current document state because the characters “cd” have been deleted. The user starts to process temporary non-selective undo. After undoing the four operations, the document state goes back to the state just after O_3 is performed. In this state, “cd” appears in the document. After the user selects the goal undo region exactly covering “cd” (Goal Undo Region in Figure 5), the user finishes this undo.

Now, let’s take a look how the agent implements the user’s requirements. First, the agent checks if O_4 is an effective operation. Since the region of O_4 partly falls in the undo region, O_4 is an effective operation and selected. According to region transformation, after that, the undo region is empty. And then the agent continues to check the rest of operations. However, O_5, O_6 and O_7 are not the effective operations, because their regions do not fall in the undo region. So the agent starts to perform a sequence of actual undo procedures. Since O_4 partly falls in the undo region, only the part of O_4 that the undo region covers is undone. O_4 is split to two operations, O_4^1 = delete(“cd”,2) and O_4^2 = delete(“ef”,2). After the inverse operation of O_4^1, insert(“cd”,2), is transposed against O_4^2, O_5, O_6 and O_7, it is performed in the current document to succeed recovering “cd”. Meantime, $O_4^{2'}$ = delete(“ef”,4), O_5' = insert(“mn”,4), O_6' = insert(“op”,6) and O_7' = delete(“mn”,4). Finally, the operation history is updated to be:

$$OH = [O_1, O_2, O_3, O_4*, O_4^1, \overline{O_4^1}, O_4^{2'}, O_5', O_6', O_7']$$

The final result satisfies the user’s expectation. The document state becomes S_8 in Figure 5.

REGIONAL UNDO INTERPRETATION

According to Abowd and Dix [1], undo facility is nothing more than a error recovery mechanism to support user intention. There are generally two ways to recover from erroneous states, namely, backward error recovery and forward error recovery. Users usually resort to forward error recovery by performing the provided *do* operations directly on the current document state, if the provided *undo* (or backward error recovery mechanism) are too awkward to use [1]. However, in forward error recovery the user must be able to recall what the "right" state was and then restore the lost information.

Backward error recovery means that, to rectify an error, the user first goes back to the state immediately prior to performing the erroneous operation and then redo all subsequent operations not related to the error from that state. Linear (or non-selective) undo is actually based on this idea. To undo an error, linear undo undoes all operations between the current document state and the state just prior to the error. Hence linear undo can always successfully go back to any previous state. Meanwhile, linear undo does not need to track dependencies among operations, because no operations are skipped in the undo procedure.

However, since linear undo undoes other operations which are not associated with seeking the user's intention, some correct contents are lost. Non-selective undo thus reaches the user's intention of error recovery at the loss of useful work. Selective undo just overcomes this limitation. However, how to select the desired operations to undo remains unsolved yet in text editors. As analyzed in the introduction, visualizing the operation history does not seem to help in text editing.

Regional undo, as a special approach to implementing selective undo, is also a backward error recovery mechanism. So regional undo also needs to undo the desired operations to recover from earlier errors. Fortunately, our approach does not put the heavy burden of looking for the desired operations on the user's shoulders. When the contents to be recovered are known, the system automatically traces all the effective operations. In other words, the user does not need to understand which operations should be undone to achieve his intention or error recovery. They only need to select the region on some document state that can exactly cover the contents to be recovered. Since the concept of temporary undo is introduced in our work, searching the right region can be performed incrementally by the user and the process is easy to understand and manipulate.

Therefore, if we say existing undo mechanisms are operation-based, regional undo can be referred to as content-based. In regional undo, the system undoes all effective operations between the current document state and the document state in which the contents to be recovered appear. Regional undo is always able to successfully recover the desired contents without losing other works, given that the user can find them in some previous state. Meanwhile, the need of tracking dependencies among operations does not arise, because no effective operation between the goal state and the current document state is skipped. In fact, linear undo can be considered as a special case of regional undo when the whole document is treated as one region.

Furthermore, since the semantics of undo is now determined by the contents to be recovered, the original operations are not considered as atomic units any more. Operations splitting and combination thus become necessary. Fortunately, the system takes the responsibility. They are not visible to the users. This is a key difference from existing approaches which always assume that undoing the original operations as a whole can satisfy the user's intention.

COMPARISON TO RELATED WORKS

DistEdit [8] and REDUCE [10] are two text editors that support selective undo and their solutions are also based on operational transformation. Although they are multiuser editors, the same ideas can be reused in single-user editors with less technical challenges. However, they did not address the problem of how the user can select operations to undo from the operation history.

In DistEdit, when undoing an operation O, it is first transformed against all operations in the history that were performed later than it, resulting in O'. Then its inverse $\overline{O'}$ is executed. However, due to the dependencies between operations, such transformations cannot always be successful. For example, a user first inserts a string "abcd" (O_1) and then deletes "ab" (O_2). Now the user wants to selectively undo O_1. However, since the later operation O_2 depends on O_1, transformation of O_1 against O_2 does not make sense at all. REDUCE solves this problem in a way similar to our approach, as discussed in the fifth section. However, REDUCE does not support partial undo as in our work.

According to Berlage [2], the selective undo models in DistEdit and REDUCE are based on a script interpretation, in which undoing an isolated operation is equivalent to skipping this operation and redoing all subsequent operations. To predict the effect of undo, the user must know the history between the operation to undo and the current state. Berlage proposes a direct selective undo model to address this problem, which only depends on the operation to undo and the current state. However, as noted by the same author [2], direct selective undo is based on a so-called stable execution property that is not always satisfiable in the text editing domain. The reason is that the content of a text document is simply regarded as a stream of characters [2]. With characters being inserted and deleted over time, the reinterpretation of the original operations is not always obvious or well defined. Therefore, direct selective undo is more suitable for graphics editors. In text editors, operational transformation can effectively solve such a problem of floating references [2].

Although regional undo is still based on the script interpretation, our approach eliminates its previous limitation of requiring the user to understand the history between the operation to undo and the current document state. The reason is that region transformation as a filter automatically selects the related operations so that the user does not need to pick them up manually.

Edwards et al. [3] and Meng et al. [6] both demonstrated a sense of regional undo in the graphics editing domain. However, regions (or segment in their terminology) in [3] are predefined in the beginning of interaction and are static. The history searching and grouping facility in [6] is effectively able to define dynamic object groups to perform undo on. The applicability of these two approaches seems limited to graphics editors where objects are clearly defined. In the text editing domain of our work, however, segments are usually not defined statically because their ranges change as contents are inserted or deleted over time. Secondly, the boundaries of objects are not well-defined in text editing and are subject to different interpretations of specific applications. Thirdly, our approach allows the user to partly undo a previous operation while it is not supported in these two systems.

CONCLUSION AND FUTURE DIRECTIONS

In this paper we motivated and described a novel technique for error recovery in the context of single-user text editing. It is the first systematical treatment of regional undo in the literature to the best of our knowledge. The presented work makes the following technical contributions. It supports undo based on arbitrary regions rather than the entire document and automates the selection of operations in selective undo at the system level rather than requiring the user to manually select operations. Therefore our approach effectively moves the burden of selective undo from locating operations in the operation history to locating regions in some concrete document state. Regions are often augmented by mnemonic audio and visual cues in modern document authoring and management tools [9]. The cognitive overhead of identifying regions is thus expected to be significantly lower than directly reconstructing context from the cryptic representation of operations in existing works. Hence the presented approach of regional undo can potentially improve the usability of selective undo as a backward error recovery mechanism. In future research, we plan to conduct usability studies to evaluate the technology and to extend it in other domains such as graphics editors and multiuser interactive applications.

REFERENCES

1. Gregory D Abowd and Alan J Dix. Giving undo attention. In *Interactive with Computers*, pages 317–342, 1992.

2. Thomas Berlage. A selective undo mechanism for graphical user interfaces based on command objects. In *ACM Transactions on Computer-Human Interaction, Vol.1, No.3*, 1994.

3. W.Keith Edwards, Takeo Igarashi, Anthony LaMarca, and Elizabeth D. Mynatt. A temporal model for multilevel undo and redo. In *User Interface Software and Technology*, 2000.

4. Clarence A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *ACM SIGMOD'89 preceedings*, pages 399–407, Portland Oregon, 1989.

5. David Kurlander and Steven Feiner. A history-based macro by example system. In *User Interface Software and Technology*, 1992.

6. Chii Meng, Motohiro Yasue, Atsumi Imamiya, and Xiaoyang Mao. Visualizing histories for selective undo and redo. In *Third Asian Pacific Computer & Human Interaction*, pages 459–464, 1998.

7. Brad A. Myers and David S. Kosbie. Reusable hierarchical command objects. In *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground*, pages 260–267, 1996.

8. Atul Prakash and Michael J. Knister. A framework for undoing actions in collaborative systems. In *ACM Transactions on Computer-Human Interaction*, volume 1, December 1994.

9. George Robertson, Mary Czerwinski, and Kevin Larson. Data mountain: Using spatial memory for document management. In *User Interface Software and Technology*, pages 153–162, 1998.

10. Chengzheng Sun. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction*, 9(4):309–361, December 2002.