Generalized Infinite Undo and Speculative User Interfaces

Alejandro López-Ortiz
School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
alopez-o@uwaterloo.ca

Abstract

We study the potential benefits of a computational model supporting an unlimited number of undo-like operations. We argue that, when implemented to its full generality this results in a novel mode of human-computer interaction in which the user session is not linear in time. The model proposed is also more resilient in the presence of user or computer errors. This resiliency can be exploited by a speculative user interface (SUI) which guesses user intentions. If the prediction is incorrect, the generalized infinite undo facility provides the ability to extemporaneously roll back the misprediction. We observe that this model is only know becoming a realistic possibility due to the current surplus of available CPU cycles on a modern desktop.

1 Foreword

This report is a position paper exploring future trends in computing. We propose a new model and speculate about its possible applications. The claims made are intended to as a guide for future research —by the author and others— that might settle some of the questions posed.

2 Introduction

This work is motivated by the observation that modern desktops have a surplus of computational power available which could be used to support novel models of interaction with users.

Revisiting the last fifty years there seem to have been certain previous periods during which computational power has exceeded the day-to-day use of computers. Yet in every instance, new software applications eventually arose to take advantage of the newly gained computational power. Along these lines, we have seen the appearance of spreadsheets, WYSIWYG word processors, GUI interfaces for operating systems, graphic intensive web browsers, always-active spell checkers and digital video, to name a few.

Over the last few years, we have once again witnessed an increase in the amount of CPU power available that apparently has not been matched by application demands. Indeed, the current bottlenecks for the most prevalent uses are generally last-mile bandwidth, disk space and CPU-bus bandwidth (in that order) rather than actual CPU availability.

In this note we propose a model for generalized infinite undo, that changes the linear nature of the human-computer interaction while allowing a richer, more speculative interaction with the user interface. This model is resource intensive which might explain why it has not been implemented in the past.

3 Name completion, automated spell checking and other agents

Over the last fifteen years, several basic computer agents attempting to simplify the interaction between humans and computers have been widely deployed. These agents range from basic filename completion, first made available in the Unix Tenex shell, and which is now relatively commonplace, to the (in)famous *Bob* and *clip-it* of Microsoft Word, which are commonly described as "annoying". On the other hand, there has been comparatively a quiet acceptance of the automated spell checker of Microsoft Word.

From the experiences described above, and using annecdotal evidence as a guide, it seems that agents are likelier to succeed if they are (a) nonobtrusive (b) their action is easily reversible and (c) often triggered at the behest of the user.

We posit that a model allowing a generalized form of infinite undo would allow user agents to be more speculative in its behaviour if it can easily be reversed, while at the same time freeing the user from the linearity of the current human computer interaction model.

To illustrate, consider a user renaming a sequence of files named `aa.1, aa.2, aa.3, ..., aa.99` to `bb.1, bb.2, bb.3,..., bb.99`. This would generally require the user to manually rename each file or to write a small program that does this. On the other hand after having seen user initiated rename command from `a.N` to `b.N` for $N = 1$ and $N = 2$, an speculative user interface (SUI) would conjecture that the user intends to rename the entire sequence. This speculative behaviour can either be triggered automatically or at the request of the user, depending on the user preferences. In either case, the SUI would then proceed to rename the files and inform the user of the speculative action taken, e.g. in the case our example:

> renaming files: `aa.3 to bb.3, aa.4 to bb.4, ..., aa.99 to bb.99`
> `DONE!`

The user could then either silently accept (the SUI derives agreement from no action as to reduce the amount of obtrusiveness), or alternatively if the speculative action is incorrect the user would issue an undo command.

Speculative behaviour might also lead to regret at a latter time, when the user has already agreed to a certain action and only later realizes that it should not have been performed. Pressumably the user would not wish to reverse all other actions that have since taken place, yet most modern undo-redo systems provide this facility alone. In contrast a generalized undo facility would provide access to the state of the system at any arbitrary point in time and have a single action reversed in an efficient manner.

4 Database rollbacks and word processing undos

Note that a limited undo functionaliy is already available in some computer settings. In this section we discuss three notable cases. First are database rollbacks which allow the user to restore the state of the database to that recoreded at an earlier time. Depending on the application, this previous state might be an arbitrary one or one of a series of checkpoints or snapshots. Traditionally, databases supporting unlimited undos have a snapshot in time from which other versions can be retrieved using the update log, while checkpoint databases limit themselves to retrieving the state of the database at the time of the snapshot.

Second is the limited undo is that provided by advanced word processors and text editors, such as Microsoft Word and Emacs. In these, the user can reverse the effect of any operation dealing with the text, usually back to the last time the file was saved.

Both the database and word processing model have an implicit concept of linear time, in which by undoing some changes time flows in reverse to obtain a previously reached state, with the "newer" versions becoming unaccesible. This is a linear model of time flow.

A third example is the one provided by version control systems such as RCS and CVS. In this setting, as in the logged database case, the entire history of updates is stored, making it possible to recover any given version of a document in the past. Both the CVS and database update are tailored to unfrequent retrievals of old versions and do not support a straightforward branching of the version tree or reversal of a single action in time. That is, assume that at a point in time the name of an object and its methods in a Java file were changed by a straightforward search and replace operation. Thereafter other changes not involving the name of the object have taken place. In this case substantial effort would be required to roll back the change while preserving other changes thereafter.

A true infinite undo facility would allow the reversal of the search/replace operation while preserving other changes. As well, the model should ideally preserve both the version before and after the undo so that the user can verify if the requested action did indeed take effect as expected or if the "undo" itself needs to be undone (redo).

5 Nonlinear time

In this section we consider natural examples where users are likely to benefit from non-linear access to the timeline.

- The history facility of a web navigation session provides a natural example of a non-linear navigation in time. Currently, if a user presses the back button a few pages and then follows up a new link from that page, the actions that occured after that page disappear from the linear presentation of the navigation timeline. To be more precise, if the user has navigated pages A, B, C, D, E in the current session, he/she can move along that timeline using the `back` and `forward` buttons on the web browser. If the user navigates back to page, say C and then selects a new link D' and from then E' and F', the new timeline is A, B, C, D', E', F'. Pages D and E are no longer accessible through the back and forward mechanism.

- A CVS version control system the user might want to recover an old version of a document or program and develop it in parallel, or perhaps fix a bug in many versions at once by going back in time and performing a fix at that point in time as it had always been there. In this case we need the ability to navigate to a time back in the past an effect a change that will automatically propagate to the future.

- Similarly, cooperative tasks such as editing of a large document or programming project result in the creation of parallel time lines. For example programmers A and B check out a file, each implements a sequence of changes thus creating their own private time-line and then later on those changes are merged. In this case not only does time branches, as we have seen before, but it also merges back later on.

- The file system itself is a natural candidate for an infinite undo facility. Modern operating systems provide by default a limited undo facility by means of a trash can or recycling bin to which files are removed to. In addition if a regular backup process is in place the user might be able to retrieve older versions, within the resolution provided by the backup process. However this does not provide a resolution fine enough to undo any specific arbitrary operation or support a nonlinear view of time.

6 Technical Challenges

An infinite undo facility can be implemented by simply preserving a copy of the state of the machine at every specific point in time. This would be very inefficient in terms of both storage requirements and the delay in the write out of the state to persistent storage.

Alternatively a syntetic representation of the past can be accomplished in much less space. For example, databases achieve this by storing the entire data sparsely and interpolating state via the transaction log. Similarly CVS repositories store a sequence of "diff" files. Both of these models operate under the assumption that reconstruction of the past is a rather unfrequent event.

A more commonly used undo facility *might* require more efficient implementation of these operations. In fairness, it is difficult to estimate the actual rate of update. Would people come to rely on an undo facility if one was to be provided? or would it remain a temporary, last resort tool, to be accessed rarely? (consider for example the last time that one accessed the windows recycle bin).

In any event a substantial penalty in the performance of the system is to be expected. In exchange, the user gets (a) a computing facility with almost no data loss and (b) the ability to effect changes in a post-facto basis in an efficient manner.

Another interesting aspect is the definition of an algebra of operations that allow for a non-linear timeline. What is the result of modifying an element in a record that no longer exists due to a previous undo?

As it is, modern desktops already require search utilities to navigate stored data. Persistence would only increase the amount of data available hence the ability to search across time for a specific target is important. Novel algorithms are required to support efficient searches across time.

7 Conclusion

The examples provided justify the need for an infinite undo facility. Depending on the specific application, this might be readily be implemented or could possibly require a substantial amounts of CPU power and permanent storage (hard drive). In either case we posit that the changes needed are within the ability of modern computer power.

References

[1] Alan Dearly, Quintin I. Cutts and Graham N.C. Kirby. Browsing, Grazing and Bibbling Persistent data Structures, *Proceedings of the Third International Persistent Object Systems Workshop, 1989, Workshops in Computing, Springer, pp. 56-69.*

[2] A. Cypher, ed. *Watch What I Do: Programming by Demonstration.* MIT Press, 1993.

[3] B.D. Davison and H. Hirsh. Toward an adaptive command line interface. *In Proceedings of the Seventh International Conference on Human-Computer Interaction.* Elsevier, 1997.

[4] H. Hirsh and B. D. Davison. An adaptive UNIX command-line assistant. *In Proceedings of the First International Conference on Autonomous Agents.* ACM Press, 1997.