Title:
Detecting Local Undo Conflicts in Multi-User CAD

Authors:
David J. French, davidfrench11@gmail.com, Brigham Young University
Scot Wilcox, wilcox.scot@gmail.com, Brigham Young University
Kevin Tew, kevin_tew@byu.edu, Brigham Young University
Edward Red, ered@byu.edu, Brigham Young University

Keywords:
Multi-user, CAD, Undo, Conflict, Synchronous, Collaboration

DOI: 10.14733/cadconfP.2015.407-410

Introduction:

Multi-user computer-aided design (CAD) is an emerging technology that promises to facilitate collaboration, enhance product quality, and reduce product development lead times by allowing multiple engineers to work on the same design at the same time [5]. The Brigham Young University (BYU) site of the NSF Center for e-Design has developed advanced multi-user CAD prototypes that have begun to demonstrate the advantages of this technology.

There are many new challenges to address to adequately support multi-user CAD. Several research efforts have already been conducted to address some of these challenges: Jing et al. and Liao et al. have studied identification (naming) of CAD features [6-7]; Hepworth has studied feature reservation for conflict avoidance [3]; and Hepworth et al. and Cai et al. have studied client model consistency [2],[4]. Undo/redo in multi-user CAD is an important challenge that needs to be addressed more completely.

Abowd and Dix claimed that "Few people would argue about the importance of undo," [1]. A large engineering company recently used an analytics tool provided by BYU to track which buttons their designers clicked the most in single-user CAD. They reported to the researchers that their engineers clicked the Undo button an average of four times more than any other single button in the CAD application. This validates the importance of undo in single-user CAD, and it is likely that undo will be very important in multi-user CAD as well.

As in other multi-user applications, conflicts can occur during local undo in multi-user CAD due to the dependencies between commands from different users. Due to the high number and complexity of dependencies between CAD features, these conflicts are more likely to occur and are more difficult to detect than conflicts in many other types of multi-user applications.

A conflict occurs in local multi-user undo when:

- the undo does not or cannot provide the expected result due to commands that were performed by other users after the command that is being undone, or
- the undo affects the results of commands that were performed by other users after the command that is being undone. So, undo can cause unintended consequences for either the local user or for other users.

Consider the following history list, where A_i refers to commands performed by User A, and B_i refers to commands performed by other users:

$$A_1\ B_1\ A_2\ B_2\ B_3$$

When User A performs an undo command, local undo will attempt to revert the A_2 command. If A_2 is independent of B_2 and B_3, then no conflicts will occur. However, if there are dependencies between A_2 and either B_2 and B_3, then a conflict may occur.

A method and set of principles have been developed for detecting when an undo command in multi-user CAD conflicts with commands performed by other users after the command that is being undone. This method checks for parametric dependencies (both child and parent dependencies) between CAD features affected by the undo command and features affected by more recent commands from other users. It can perform this dependency check prior to undoing the command. This method catches syntactic conflicts and some potential semantic conflicts that occur during local undo/redo in multi-user CAD.

A method for detecting conflicts in multi-user CAD before an undo command is performed will provide at least the following three benefits:

1. It will detect syntactic conflicts and some possible semantic conflicts before an undo or redo command is allowed. This will provide an opportunity to prevent or resolve the conflict.

2. It provides a way to inform the local user why their undo cannot or should not be executed and which other user performed the conflicting command. This allows the local user to collaborate with the user that performed the conflicting command to resolve the conflict. For example, the local user could ask the other user to undo their conflicting command so the local user's undo command can succeed, or both users could collaborate using standard (non-undo) commands to resolve the conflict.

3. It provides a way to inform the local user when another method or mechanism may be more suitable than local undo for accomplishing the user's intent. Examples of such alternative methods and mechanisms include standard (non-undo) commands; manual conflict resolution tools (commonly known as "diff" tools); and other types of undo mechanisms such as selective, regional, or global undo that could allow the local user to undo other users' conflicting commands before undoing their own command.

Main Idea:

A general classification of conflicts that can occur during local multi-user undo/redo has been developed. These conflicts can be generalized by considering the Pre-Operation State and Post-Operation State of a feature (see Tab. 1). The Pre-Operation State is the state a feature is in prior to any operation, including prior to an undo operation. The Post-Operation State of a feature is the state a feature is intended to be in after an operation occurs, including after an undo operation occurs. For example, when undoing a delete operation, the undo operation is really "re-creating" the feature. Thus the Pre-Operation state is null (the feature does not exist in its deleted state), and the Post-Operation State is not null (the deletion is being undone, thus creating the feature again). If the Pre-Operation State is not null (if undo is deleting or editing a feature), there are two main types of conflicts that can occur:

- Type A: Self-Self conflicts (Fig. 1, (left))
- Type B: Self-Child conflicts (Fig. 1 (middle))

If the Post-Operation State is not null there is one main type of conflict that can occur:

- Type C: Parent-Self conflicts (Fig. 1 (right))

To detect Type A (Self-Self) conflicts, we store a single Operation Number (ON) on the client that tags features with an integer each time they are created or edited by someone. The ON increments on every command applied to the model from either the local user or other users. For example, if we start with an empty part, and then create two features, the first feature would have an ON of 1 and the second feature would have an ON of 2. If another user edited the first feature, it would then have an ON of 3. We store that number in the undo operation that gets made when the feature is created, so that when the user tries to undo, we can compare the undo operation's number to the feature's current Operation Number to see if the local user was the last person to create or edit that feature. In this

fashion we know when a feature involved in an undo dependency has been changed by another user, which might cause a conflict,

		Post-Operation State	
		Deleted (null)	Exists (not null)
Pre-Operation State	Not yet created (null)	n/a	Create
	Exists (not null)	Delete	Edit

Tab. 1: Pre- and Post-Operation States of a feature.

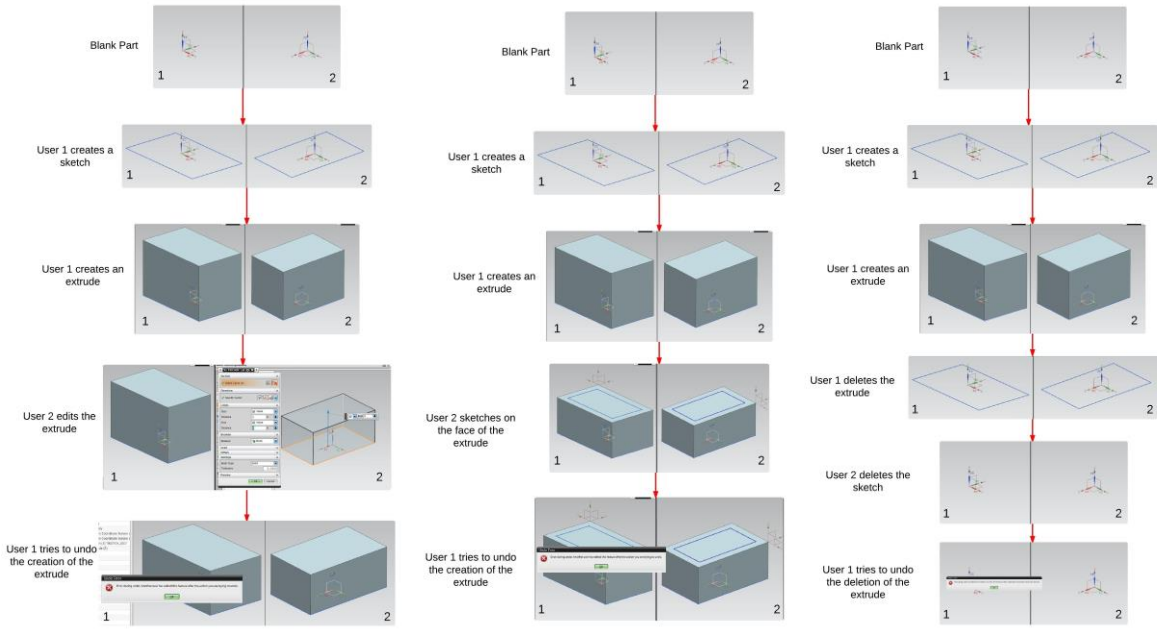

Fig. 1: Examples of Type A (left), Type B (middle), and Type C (right) undo conflicts.

To detect Type B (Self-Child) conflicts, we compare the Operation Number with that of the child features of the feature to be undone. If any are greater than the parent ON, we know that someone has edited that child feature since the local user's original action was performed. In this case, a feature's children are known since the undo dependency check happens before the undo is allowed to be performed, and the dependent children can be queried from the API directly.

To detect Type C (Parent-Self) conflicts, we have to store a list of parent features each time the Pre-Operation State is not null. This list is updated every time a state is "left". Before an undo occurs to go back to the prior Pre-Operation State, the model is checked to ensure that (a) the parent features still exist, and (b) the parent features have not been edited by other users since the previous state was left. In order to make sure (b) is true, we compare each parent feature's ON with the undo operation's saved ON and make sure they are all less than the saved ON.

This undo conflict detection method was implemented in BYU's NXConnect prototype. The prototype succeeded at detecting all of the conflict types described above. The screenshots shown in Fig. 1 demonstrate the NXConnect prototype responding appropriately to each conflict type.

The method given is applicable to redo as well as undo. Following an undo command, it is possible for other users to perform commands that will conflict with the local user's subsequent redo command. The conflict detection method and system response described for undo can be similarly applied to redo.

Conclusions:

Detecting conflicts in local undo in multi-user CAD prevents model corruption and provides an opportunity for the users involved in the conflict to collaborate to resolve conflicts. This is an important technical challenge to address in multi-user CAD since undo is such a commonly used command, and since detecting conflicts in local multi-user undo is a prerequisite for resolving those conflicts.

A classification of conflicts that can occur during local undo in multi-user CAD has been developed, and a method for detecting and warning about these conflicts has been provided. This method has been successfully implemented and tested in BYU's NXConnect multi-user CAD prototype.

Future research could be conducted on multi-user design strategies for preventing conflicts, such as assigning users to work in separate, independent regions of the model. Other future research could study enhanced methods for conflict resolution, such as providing a "diff" tool to allow the user to select which version of the model they want to keep when multiple, conflicting results are possible from an undo command.

Acknowledgements:

The authors would like to thank the NSF Center for e-Design and its industry sponsors for funding this research.

References:

[1] Abowd, G. D.; Dix, A. J.: Giving undo attention, Interacting with Computers, 4(3), 1992, 317–342. http://dx.doi.org/10.1016/0953-5438(92)90021-7

[2] Cai, X.; He, F.; Jing, S.; Liu, H.: A consistency and awareness approach to naming merged faces in collaborative solid modeling, International Conference on Computer Supported Cooperative Work in Design, 2008, 803–807. http://dx.doi.org/10.1109/CSCWD.2008.4537082

[3] Hepworth, A.; DeFigueiredo, B.; Shumway, D.; Fronk, N.; Jensen, C. G.: Semantic conflict reduction through automated feature reservation in multi-user computer-aided design, International Conference on Collaboration Technologies and Systems, 2014, 56–63. http://dx.doi.org/10.1109/CTS.2014.6867542

[4] Hepworth, A.; Tew, K.; Trent, M.; Ricks, D.; Jensen, C. G.; Red, W. E.: Model Consistency and Conflict Resolution With Data Preservation in Multi-User Computer Aided Design, Journal of Computing and Information Science in Engineering, 14(2), 2014, 021008. http://dx.doi.org/10.1115/1.4026553

[5] Hepworth, A. I.; Tew, K.; Thomas, N.; Mark, B.; Jensen, C. G.: Automated Conflict Avoidance in Multi-user CAD, Computer-Aided Design & Applications, 11(2), 2013, 141–152. http://dx.doi.org/10.1080/16864360.2014.846070

[6] Jing, S.; He, F.; Liu, H.; Liao, B.: Conflict Analysis in Replicated Collaborative Solid Modeling Systems, Computer Graphics and Virtual Reality, 2006, 175–181.

[7] Liao, B.; He, F.; Jing, S.: Replicated collaborative solid modeling and naming problems, Conference on Computer Aided Design and Computer Graphics, 2005, 3–8. http://dx.doi.org/10.1109/CAD-CG.2005.72