COMPUTER-AIDED DESIGN & APPLICATIONS, 2016
VOL. 13, NO. 6, 760–767
http://dx.doi.org/10.1080/16864360.2016.1168217

Detecting local undo conflicts in multi-user CAD

David J. French, Scot Wilcox, Kevin Tew and Ed Red

Brigham Young University

ABSTRACT
Multi-user computer-aided design (CAD) is an emerging technology that facilitates collaboration by allowing users to work on the same design at the same time. This paper provides a classification of the types of conflicts that can occur during the use of local undo in multi-user CAD, and describes a method for detecting those conflicts. This method can be used to prevent and warn the user about undo conflicts as well as provide users with information about the cause of the conflict so they can collaborate to resolve it. This method has been successfully implemented and tested in Brigham Young University's NXConnect multi-user CAD prototype.

KEYWORDS
Multi-user; CAD; Undo; Conflict; Synchronous; Collaboration

1. 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 [11]. 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 [12,13]; Hepworth has studied feature reservation for conflict avoidance [9]; and Hepworth et al. and Cai et al. have studied client model consistency [3],[10]. 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.

A method and set of principles has 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.

This dependency check is made faster by storing metadata about features, so that the undo command does not need to check its dependencies with every other more recent command from other users. To facilitate this method, some metadata is stored about features, such as the last time that features have been edited, and by whom each feature was last edited. Some metadata is also stored in each command regarding the previous and new state of each command's feature.

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.

CONTACT David J. French davidfrench11@gmail.com

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.

2. Background

2.1. Undo types and intent

Abowd and Dix describe two types of undo in multi-user applications: (1) local undo, where the local user's actions are reverted, and (2) global undo, where the actions of all users are undone [1]. Prakash and Knister describe other types of undo as well, such as regional undo or selective (arbitrary) undo [16].

Yang stated that "undo should be seen as an intention of the user, not an aspect of system functionality," [17]. Chen [4] states that single-user undo is most commonly used for the following user intents:

1. Recovering from an unintended or incorrect operation
2. Learning a new feature by trial and error
3. Exploring alternatives

Prakash and Knister found that users typically use local undo more than global undo in multi-user software due to lack of predictability and awareness of which command global undo is going to undo [16]. Local undo also accomplishes the intent of single-user undo better than global undo does.

2.2. Command pattern

The Command Pattern is a software design pattern that is very useful for supporting undo in both single-user and multi-user CAD. In the Command Pattern, a historical list of all undoable commands performed on the model is stored [2],[6]. A representation of a command history list is shown in Fig. 1 (from Berlage [2]). In Fig. 1, Q_i represents the state of the CAD model at a certain time, and c_i represents a specific command object. According to Meyer, "A command object represents the information needed to execute a user-requested operation and, if undoing is supported, cancel it," [15]. One way that a command object can store this information is by storing the state of the CAD feature before and after the feature edited (or created or deleted). In other words, command c_i stores the states Q_{i-1} and Q_i. An example command object is shown in Fig. 2. When c_i is performed, the state Q_i is established in the model. When c_i is undone, the state Q_{i-1} is established.

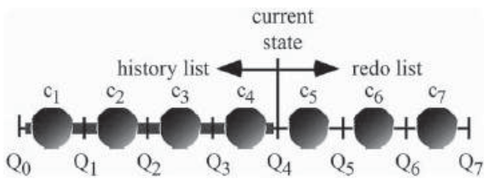


Figure 1. Operation history list (from Berlage [2]).

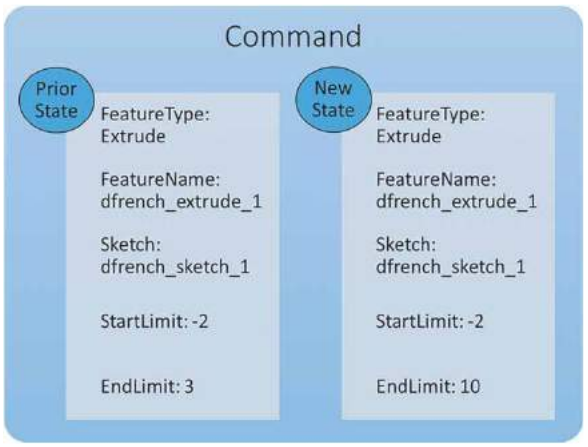


Figure 2. An example of a command object.

2.3. Linear vs. nonlinear undo

Berlage states that "undo models can be divided into two categories: linear undo, which uses a chain of commands where only the previous and next command can be undone or redone; and nonlinear undo, which allows the user to undo arbitrary actions," [2]. Since commands from each user are usually mixed in the command history list in multi-user applications, local undo requires nonlinear undo to be supported.

2.4. CAD feature dependencies

In CAD, features can have multiple parent features (features it references) and multiple child features (features that reference it). Fig. 3 (left) shows a sketch (at the base of the model) that has multiple child and grandchild features. Fig. 3 (right) shows an extrude that has multiple parent and grandparent features.

2.5. Undo conflicts in multi-user CAD

As in other multi-user systems, conflicts can occur during local undo in multi-user CAD due to its nonlinear nature and due to 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.

2.5.1. Syntactic and semantic conflicts

There are two types of conflicts that can occur: (1) syntactic conflicts, and (2) semantic conflicts. Contero et al. suggest that syntactic deficiencies in CAD are deficiencies in the mathematical validity of the model (e.g. invalid CAD features), and semantic deficiencies are aspects of the model that fall short of the user's design intent, despite being mathematically valid [7].

There are several types of syntactic and semantic conflicts that can occur during local undo in multi-user CAD. These conflicts need to be understood and addressed to preserve the integrity of model data and to preserve design intent during multi-user design collaboration.

2.6. Previous work

Multi-user conflicts and undo conflicts have been studied extensively for multi-user text editing [4, 16, 17]. Many multi-user text editing conflicts can be resolved automatically using a method known as operational transform. Significant conflict scenarios not handled by operational transform in multi-user text editing happen when two commands operate on the same position or adjacent positions [16]. Conflict detection is as simple as checking if two commands operate on the same or adjacent characters.

Conflict detection and resolution is more difficult in multi-user CAD than in multi-user text editing, because each CAD feature can and frequently does have multiple parent features and multiple child features. Some research has been conducted for detecting conflicts in multi-user CAD.

Liu et al. compare dependence between two concurrent commands based on whether the commands create Boolean operations that intersect one another or not [14]. This requires a CAD kernel-level comparison between the results of the two commands. When checking dependencies between the command being undone and more recent commands from other users, this would require executing the undo command and comparing the results

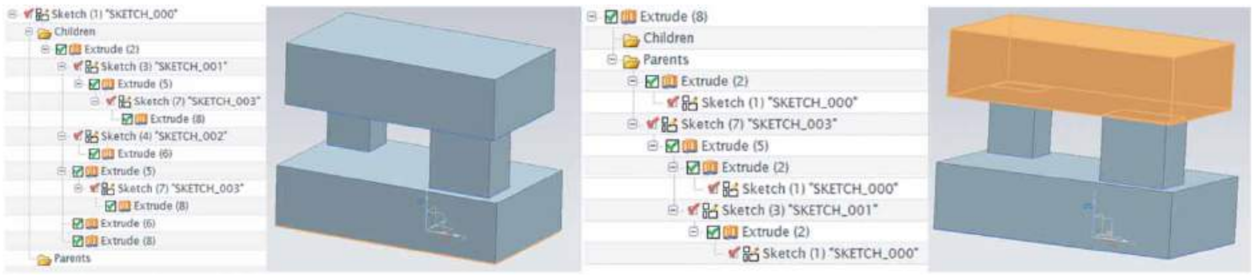


Figure 3. Child features of a sketch feature (left), and parent features of an extrude feature (right).

of that command with every other command that has been executed by other users since the command that is being undone. It would also not catch all conflicts such as parametric dependencies between features that do not directly involve a Boolean operation. For example when a sketch is created on a face, the sketch is dependent on the face even though no Boolean operation connects the sketch and the face.

Cheng et al. compare the dependence between the command being undone and recent commands from other users based on whether the recent command from another user is dependent on (i.e. a "child" of) the feature being undone [5]. This method will detect some conflicts but fails to detect if prerequisite "parent" features still exist before allowing a feature to be undone to a non-deleted state. Cheng et al.'s method also requires the undo command to be checked for dependence with every more recent command from other users. Gao et al. use the same method as Cheng et al., but extend it apply to commands that involve multiple operations [17]. The same limitations apply to Gao et al's method as apply to Cheng et al.'s.

3. Classification of undo conflicts

A classification of the types of conflicts that can occur during local multi-user undo/redo has been developed. These conflict types can be understood 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 (4 subtypes)
- Type B: Self-Child conflicts (4 subtypes; can be treated as 2 subtypes)

Table 1. Pre- and post-operation states of a feature.

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

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

- Type C: Parent-Self conflicts (4 subtypes; can be treated as 2 subtypes)

3.1. Type A (self-self) undo conflicts

Type A conflicts occur when the undo command is operating on a feature that another user has modified since the command being undone was originally performed. For example, Fig. 4 (left) walks through such a situation. Because User 2 edits the extrude that User 1 created before User 1 attempts his undo operation, a Type A conflict occurs.

3.2. Type B (self-child) undo conflicts

Type B conflicts occur when other users have performed more recent commands that create or edit features that are dependent on the feature being undone. In Type B conflicts, it may be possible for the undo to occur, but the undo may negatively affect the recent actions or intent of other users. Fig. 4 (right) shows how this can occur. Since User 2 draws a sketch on the extrude User 1 creates, the sketch is dependent on the extrude. This results in a Type B conflict when User 1 tries to undo the creation of his extrude.

3.3. Type C (parent-child) undo conflicts

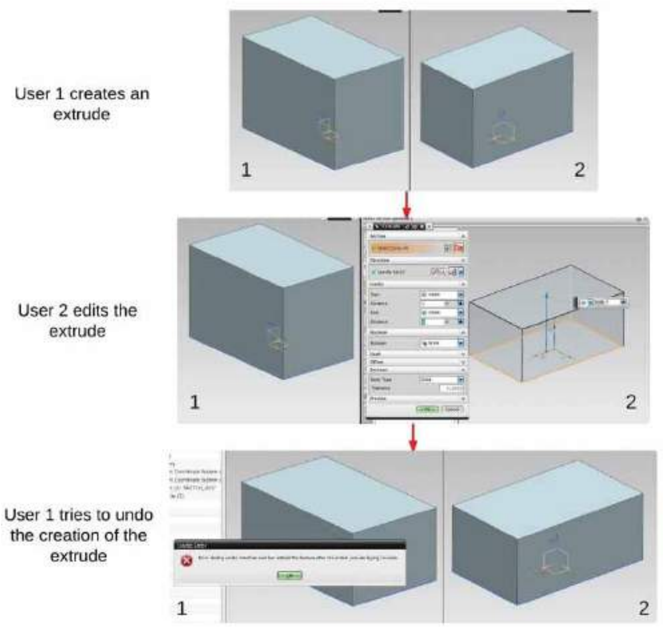
Type C conflicts occur when other users have performed commands that affect the validity or intent of the undo being performed, because the necessary parent features either no longer exist or have been modified in a way that may conflict with the intent of the undo. In Fig. 5 (left) we see a Type C conflict. When User 1 tries to undo his delete command, the original sketch his extrude was based on does not exist anymore; therefore his extrude cannot be un-deleted.

3.4. Undo conflict subtypes and suggested system responses

Table 2 describes the various subtypes of undo conflicts, their implications, and the suggested system response for each case.

4. Method for detecting undo conflicts

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


User 1 creates an extrude
User 2 edits the extrude
User 1 tries to undo the creation of the extrude
1
2

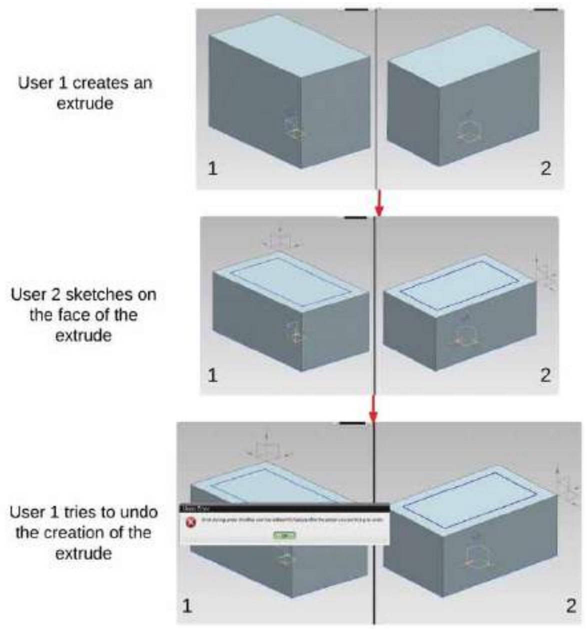
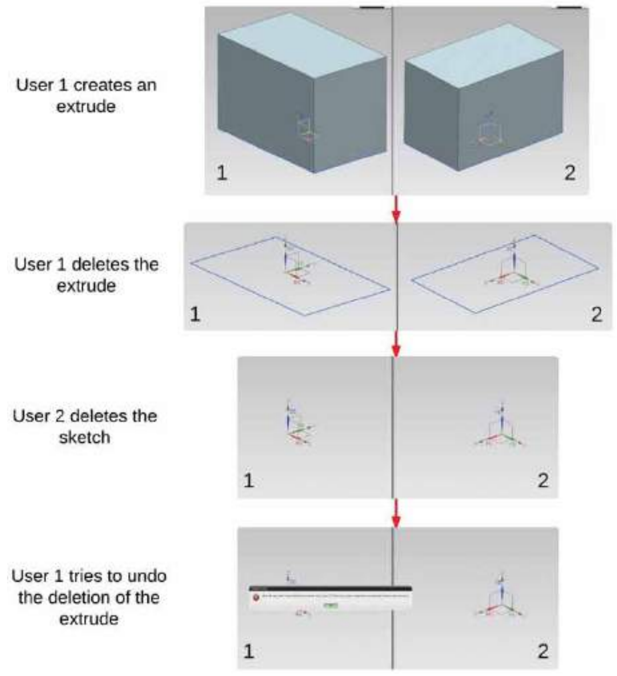

User 1 creates an extrude
User 2 sketches on the face of the extrude
User 1 tries to undo the creation of the extrude
1
2

Figure 4. Examples of Type A (left) and Type B (right) undo conflicts.


User 1 creates an extrude
User 1 deletes the extrude
User 2 deletes the sketch
User 1 tries to undo the deletion of the extrude
1
2

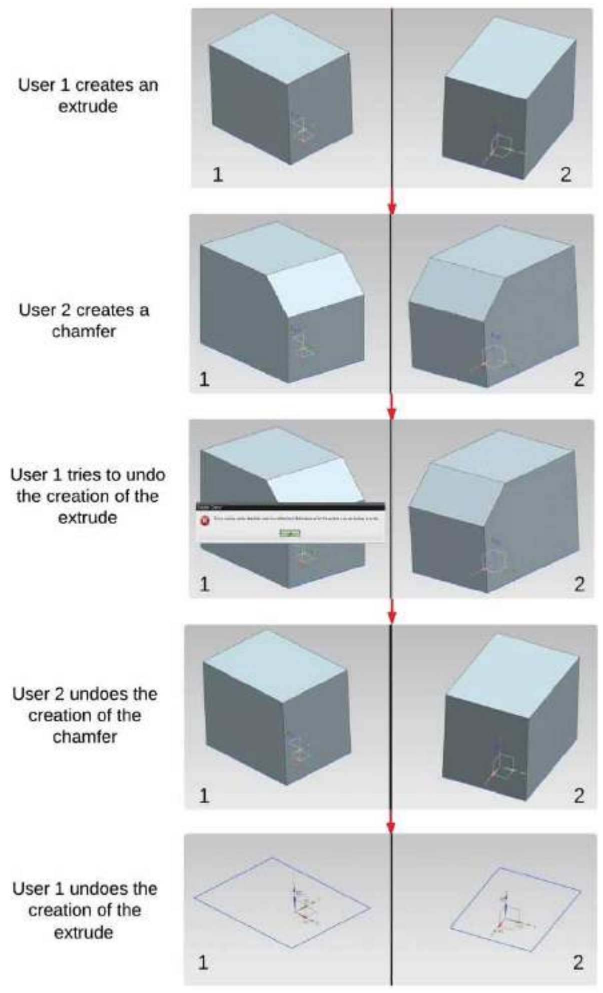

User 1 creates an extrude
User 2 creates a chamfer
User 1 tries to undo the creation of the extrude
User 2 undoes the creation of the chamfer
User 1 undoes the creation of the extrude
1
2

Figure 5. Example of a Type C (Parent-Self) undo conflict (left) and successive undo commands (right).

Table 2. Undo conflict subtypes, implications, and suggested system responses.

Type	Description	Implications	Suggested System Response
A.1	The intent of the undo operation is to delete a feature, but that feature has been deleted by another user.	The undo could be allowed since the result is the same as the other user's intent.	"The feature you are trying to undo has already been deleted."
A.2	The intent of an undo operation is to delete a feature, but that feature has been recently edited by another user.	Allowing the undo is syntactically possible; however, the deletion may interfere with the other user's intentions.	"The feature you are trying to undo has been edited by User X. Your undo operation may conflict with User X's design intent. Do you wish to proceed?"
A.3	The intent of an undo operation is to edit a feature, but that feature has been deleted by another user.	The feature could be re-created back to its pre-edit state; however, this may interfere with the other user's intentions for deleting the feature.	"The undo operation is not allowed because the feature you are trying to undo has been deleted."
A.4	The intent of an undo operation is to edit a feature, but that feature has been recently edited by another user.	The undo-edit could be allowed syntactically, but it may interfere with the other user's intentions.	"The feature you are trying to undo has been edited by User X. Your undo operation may conflict with User X's design intent. Do you wish to proceed?"
B.1/B.2	The intent of an undo operation is to delete a feature, but child features have been (B.1) created or (B.2) recently edited since the most recent command performed by the local user.	Allowing undo to delete the feature will invalidate child features.	"The undo operation is not allowed because the feature you are trying to undo has child features that have been recently created or edited by User X."
B.3/B.4	The intent of an undo operation is to edit a feature, but child features have been (B.3) created or (B.4) recently edited by another user.	Allowing undo to edit the feature may cause syntactic or semantic conflicts.	"The feature you are trying to undo has child features that have been recently created or edited by User X. Your undo operation may conflict with User X's design intent. Do you wish to proceed?"
C.1/C.2	The intent of an undo operation is to (C.1) create or (C.2) edit a feature, but one of the feature's parent features have been deleted by another user.	Without the prerequisite parent features, the undo operation cannot put the feature back to its intended state.	"The undo operation is not allowed because the feature you are trying to undo depends on parent features that have been deleted."
C.3/C.4	The intent of an undo operation is to (C.3) create or (C.4) edit a feature, but one of the feature's parent features has been recently edited by another user.	The feature may not be able to be restored to its intended state.	"The feature you are trying to undo has parent features that have been edited by User X. Your undo may or may not be successful in returning to its previous state. Do you wish to proceed?"

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.

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.

4.1. Successive undo commands by multiple users

We also allow multiple users to undo in succession. To facilitate this, we store the previous Operation Number of a feature in the undo operation, that is, the ON the feature was tagged with before the user's current ON. When undo is performed, the feature is re-tagged with its previous integer. This allows the user who operated on a feature before the local user to successfully undo their previous change, thus allowing users to collaborate to undo their operations in the reverse order of when those actions occurred. Fig. 5 (right) shows two users collaborating with successive undo commands to resolve User 1′s undo conflict. If we did not reset the Operation Number on the feature involved in the undo, then the other client will still detect that the feature has been changed even though the local user's undo command restored the feature to its prior state.

4.2. Handling multiple operations in a single user command

It is possible for a single user command to contain multiple operations. For example, a user selects three CAD features and deletes them in a single command. That user command would contain three operations (deleting each of the three features). It is possible for one of the operations in a command to fail the conflict detection check, although the other operations pass the conflict detection check. In this case, we suggest that it may be most practical to disallow the undo of all operations in that command, since the user likely expects the entire command to succeed or fail.

4.3. Depth of dependency checking

Note that since grandchild/parent or great-grandchild/parent features might have been created, deleted, or recently edited by other users, a decision needs to be made as to how many layers of feature dependencies to check. In a large CAD model with many inter-feature dependencies, it might be impractical to check all levels of child features to see if any have been newly created or edited since the user's most recent command. Our implementation only checks the first level of dependent child features.

4.4. Redo

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.

5. Results

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 Figures 4 and 5 demonstrate the NXConnect prototype responding appropriately to each conflict type.

6. Conflict prevention and resolution (future research)

The method described in this paper provides a method for detecting conflicts that would happen if an undo or redo operation was to be performed. Once the conflict is detected, the software can prevent the undo command from being performed, and the user can be given valuable information to help them resolve the conflict, or to at least understand why their undo command failed.

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.

7. Summary

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 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.

A classification of the types 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.

8. Acknowledgement

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

ORCID

David J. French http://orcid.org/0000-0002-3202-5590
Ed Red http://orcid.org/0000-0002-9321-6913

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] Berlage T.; Genau A.: From Undo to Multi-User Applications, Human Computer Interaction, 733, 2003, 213–224. http://dx.doi.org/10.1007/3-540-57312-7_70

[3] 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

[4] Chen, D.; Sun, C.: Undoing any operation in collaborative graphics editing systems, International ACM SIGGROUP Conference on Supporting Group Work, 2001, 197–206. http://dx.doi.org/10.1145/500286.500316

[5] Cheng, Y.; He, F.; Xu, B.; Han, S.; Cai, X.; Chen, Y.: A multi-user selective undo/redo approach for collaborative CAD systems, Journal of Computational Design and Engineering, 1(2), 2014, 103–115. http://dx.doi.org/10.7315/JCDE.2014.011

[6] Command Pattern, Wikipedia.org, 2014, http://en.wikipedia.org/wiki/Command_pattern. [Accessed: 11-Dec-2014].

[7] Contero, M.; Company, P.; Vila, C.; Aleixos, N.: Product data quality and collaborative engineering, IEEE Computer Graphics and Applications, 22(3), 2002, 32–42. http://dx.doi.org/10.1109/MCG.2002.999786

[8] Gao, L.; Lu, T.; Gu, N.: Supporting semantic maintenance of complex Undo operations in replicated Co-AutoCAD environments, International Conference on Computer Supported Cooperative Work in Design, 2009, 84–89. http://dx.doi.org/10.1109/CSCWD.2009.4968039

[9] 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

[10] 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

[11] 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

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

[13] 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

[14] Liu, H.; He, F.; Li, X.; Huang, Z.: A less constraint concurrency control and consistency maintenance in collaborative CAD system, International Conference on Computer Supported Cooperative Work in Design, 2010, 104–109. http://dx.doi.org/10.1109/CSCWD.2010.5471994

[15] Meyer, B.: Object-Oriented Software Construction, Prentice Hall, Santa BarbaraCA, 1988, 1073.

[16] Prakash, A.; Knister, M. J.: A framework for undoing actions in collaborative systems, ACM Transactions on Computer-Human Interaction, 1(4), 1994, 295–330. http://dx.doi.org/10.1145/198425.198427

[17] Yang, Y.: Undo support models, International Journal of Man-Machine Studies, 28(5), 1988, 457–481. http://dx.doi.org/10.1016/S0020-7373(88)80056-7