Automated Conflict Avoidance in Multi-user CAD

Ammon I. Hepworth[1], Kevin Tew[2], Thomas Nysetvold[3], Mark Bennett[4] and C. Greg Jensen[5]

[1]Brigham Young University, ammon.hepworth@byu.edu
[2]Brigham Young University, kevin_tew@byu.edu
[3]Brigham Young University, tom.nysetvold@gmail.com
[4]Brigham Young University, marktb1@gmail.com
[5]Brigham Young University, cjensen@byu.edu

ABSTRACT

The NSF Center for e-Design, Brigham Young University (BYU) site has re-architected Computer Aided Design (CAD) tools enabling multiple users to concurrently create, modify and view the same CAD part or assembly. This technology allows engineers, designers and manufacturing personnel to simultaneously contribute to the design of a part or assembly in real time, enabling parallel work environments within the CAD system. Such systems are only as robust as their methods for managing conflicts (i.e. simultaneous edits of the same feature by multiple users). A heavy-handed conflict prevention would limit collaborative freedom. This paper discusses an automated feature reservation method which prevents multiple users from simultaneously editing the same feature. The method is implemented in a commercial CAD system. Results show that this methodology prevents data inconsistency that results from feature/self conflicts. This system prevents CAD modeling conflicts, while providing an agile user experience within the collaborative environment.

Keywords: collaborative design, concurrent engineering, multi-user CAD, CAE.

1. INTRODUCTION

Today's commercial Computer Aided Design (CAD) systems are single user modeling and design environments. This is evidenced, within any engineering firm, by the number of designers, modelers, engineers, etc. huddled around a single workstation attempting to make needed changes during a critical moment in a design cycle. Or, watching as one individual spends days, weeks and months to build a complex assembly model from hundreds, thousands or millions of CAD parts. In the post-World War II era we had teams of engineers working simultaneously on the same large J-size sheet of mylar or vellum, completing the drawings in a fraction of the time it would take a single engineer. However, today only one draftsman can work inside the CAD drafting environment regardless of the size of drawing sheet. The days of concurrent parallel workflows have given way to the single user serial CAD workflow. The National Science Foundation (NSF) Center for e-Design, Brigham Young University (BYU) site is currently developing multi-user CAD tools which enable teams of users to simultaneously create, modify and view the same CAD part. This effort leverages commercial CAD system APIs to build plug-ins which extend existing CAD tools functionality to become multi-user. This allows teams of users to concurrently contribute to the design of a part in real time, enhancing collaboration and enabling a parallel work environment within the CAD system [4,14,16].

Interferences that occur when multiple users edit the same part or assembly are one of the central problems encountered in the development of simultaneous multi-user collaborative CAD. Varying simultaneous edits of the same or dependent geometric parameters may cause conflicts to arise within the model. When various users simultaneously input different values for the same entity, the distributed multi-user CAD system becomes inconsistent. For example, simultaneously editing the same height value, without conflict management, can result in a value of 10 mm on one user's workstation and a conflicting value of 2 mm on a second user's workstation. A conflict management system must be in place to ensure data consistency between distributed users.

The current literature describes two major approaches in overcoming the data conflict conundrum: optimistic and pessimistic. The optimistic approach assumes that conflicts are infrequent and are resolved only after they occur. It accepts data operations that are submitted asynchronously and uses algorithms to merge the data, detecting conflicts that occur [17]. Some examples of existing optimistic collaborative software systems are Google docs, CoWord and CoPowerPoint. Many of these systems utilize a technique called operational transform to keep data concurrent [8,19]. The optimistic approach can cause additional work for the user when they are required to manually resolve conflicts. It can also require redundant work if one completed operation is rejected in favor of another. The optimistic approach is often not optimal because it introduces overhead into the design process.

The pessimistic approach forces serialization of concurrent activities so conflicts cannot occur. It uses techniques to block concurrent access to certain data so that data stays consistent between users [14,17]. Pessimistic systems implement various levels of data blocking, ranging from locking small sections of the model, to a complete lock of the entire model allowing only a single user to edit that model at a time [14]. Pessimistic approaches limit user agility (the ability of the user to respond to change) by restricting users from contributing where and when they are needed.

A balance between the extremes of the pessimistic and optimistic approaches has the potential to minimize overhead and maximize concurrent user agility. This paper presents a hybrid approach between the extremes of purely optimistic or pessimistic conflict management. To avoid conflicts that would require manual resolution, the hybrid approach automatically places restrictions on model access (i.e. feature locks). It communicates user's design intent and avoids potential modeling conflicts through the use of intelligent visual cues and selection limitations. While this approach applies some restrictions and warnings to avoid manual merging of conflicts, they are set to a minimum to preserve an agile and uninterrupted multi-user experience. Fig. 1 compares the hybrid approach with the extreme optimistic and pessimistic approaches.

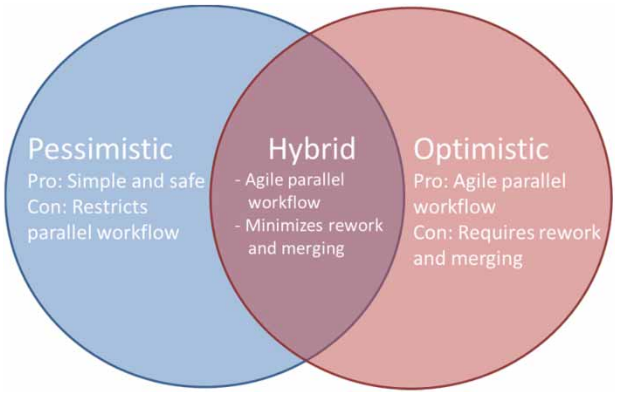

Pessimistic
Pro: Simple and safe
Con: Restricts parallel workflow
Hybrid
- Agile parallel workflow
- Minimizes rework and merging
Optimistic
Pro: Agile parallel workflow
Con: Requires rework and merging

Fig. 1: Comparison to extreme optimistic and pessimistic approaches.

2. BACKGROUND

Several methods exist for managing conflicts and data consistency in collaborative design environments. For organizational purposes, the conflict management methods are divided into the following categories: turn based, model decomposition, on demand locking, and rules based. They are listed roughly in order from the most pessimistic to most optimistic. Finally, we summarize what is lacking in these methods for an agile and uninterrupted multi-user design experience within a parametric, feature based CAD system.

2.1. Turn Based

Chan presents methods using a token based system allowing only the user who holds the token to edit the part while the other users are just observers [6]. Li utilizes a similar method that uses a turn based system for specific user functionality (i.e. viewing, deleting and adding to the part) [10]. These methods ensure that users do not make conflicting changes to the model since only one user is allowed to edit the part at a time. However, it disallows a fully parallel design workflow, causing one user to wait while another is modeling.

Bidarra et al. created a collaborative design system called WebSpiff where they assume users will coordinate their operations in a collaborative environment over phone or chat. To assist in this effort they implemented a traffic light system which visually warns users when another user is performing an operation but does not strictly lock users from making changes. Essentially, this softly reserves the entire part, warning other users about making contributions while another user is performing an operation. Their paper only discusses one type of conflict, which is where a user tries to edit a feature that no longer exists and mention that a user is notified when this occurs [2]. This method has some advantages over Chan and Li's methods because there is not a strict part lock, but they do not present enough information to determine whether the conflict management methods are sufficient for complex feature based CAD modeling.

2.2. Model Decomposition

Cera et al. developed methods to hide specific design data from certain users' view in a collaborative design environment based on each user's role. The user role controls the access of users to view certain geometry through the partitioning of 3D models. Within a partition, multi-resolution techniques are employed to obscure the geometry in that region so that it remains secure from users without the appropriate level of access [5]. Marshall presents a method of model decomposition within a collaborative CAD system that sets boundaries within a part before modeling is performed. Her method enables administrative controls to divide a model into regions or tasks. These divisions limit a user's access to other regions or tasks [14]. The pre-work required to decompose a model takes time away from the actual design work. The partitions of model decomposition reduce the agility of the design process and inhibit the flexibility of a parallel design workflow.

2.3. On Demand Locking

Bu et al. presented an approach for object locking within a collaborative graphics design environment to avoid conflicts in user's design intent, also known as semantic preservation. Their methods of locking both regions and objects on demand, giving the user an opportunity to attach design intent data or block users from making specific design changes within the region or object. They also presented a method to overcome conflicts in semantic locking operations. They successfully implemented these methods in a collaborative graphics design system called CoDesign [3,10]. Moncur also presented a method for on demand feature reservation within a commercial 3D CAD system. His method allows users to reserve a feature or group of features for a specified duration of time so that other collaborative users have limited or no access to the feature(s) [14]. These methods allow for data consistency in a manual way by requiring users to intelligently reserve features as necessary. However, his method requires users to predict when conflicts will occur, which adds some additional overhead to the design process.

2.4. Rules Based

Shen et al. introduce a collaborative drafting tool to aid in mechanical engineering design education. This tool handles conflicts by implementing authorization rules and a team manager. The authorization rules allow other users to modify one designer's entities only when they are authorized to do so. In addition, a user who makes changes to entities that he owns will automatically override any changes made by other users. The team manager acts as a team coordinator and mediator between users; however, he acts as a common designer when making changes to the model [18].

Chen et al. present their collaborative assembly modeling system called e-Assembly. This system allows multiple users to jointly build and constrain an assembly model in real time, based on coordination rules to help avoid conflicts. The rules govern who works on which link entity and allow only one collaborator to work on the same atomic component or link entity [7].

Lin et al. acknowledge that if operations in a multi-user collaborative environment are competing with each other, one of the operations has to be removed. They suggest that a method which involves eliminating user's operations through blocking/aborting is less desirable than a masking method because when such a method is employed, these operations are lost to the system and cannot be brought back when necessary. The masking method, on the other hand, maintains both resulting variations of the model when conflicts occur, but only displays the version that has the highest priority, thus masking all other variations. Since all variations are stored internally, if the operation with the highest priority is removed, the system displays the operation with the next highest priority. This masking strategy was implemented in Collaborative Genetic Software Engineering to manage tree structure constraints. They suggest that the masking method may also be applied to CAD, spreadsheets, graphical interface toolkits, and simulation systems [12].

Agustina et al. suggest that an optimistic approach using operational transform is not possible for a CAD system due to the relational complexity of the features in a CAD part; however, research by Jing et al. has shown that this may be feasible [1]. They utilize operational transforms to merge multiple compatible user operations on uniquely named topology, allowing for a less constrained multi-user interaction within a collaborative CAD system. They do not however, discuss methods for managing conflicts on multiple child features referencing the same topology or handling conflicts between incompatible features [10].

2.5. Background Summary

Turn based conflict management systems are severely limiting for a parallel design workflow because only one user can edit the part at a time. Model decomposition takes time away from the actual design because of the necessary pre-work to divide the model. It reduces the agility of the design process by forcing users to work in a confined area. On demand feature locking methods require users to predict when conflicts will occur. Future rules-based approaches have the potential to provide the least overhead and most agility for a collaborative CAD environment.

3. CONFLICT MANAGEMENT METHODS

We present an automated, rule-based conflict management system for a multi-user CAD environment. This system allows multiple users to edit the same part simultaneously, without dividing the part into single user regions or manually locking features.

3.1. Types of Conflicts in Feature Based CAD

There are two main types of conflicts in feature based CAD: syntactic and semantic conflicts. Syntactic conflicts result from multi-user operations which cause incompatible data to coexist and lead to errors in the CAD program. Semantic conflicts originate from users misunderstanding each other's design intent which lead to unexpected design results. Within a parametric feature based CAD system there are various types of potential syntactic conflicts which occur in a part. We have classified syntactic conflicts into the following categories: feature/self, parent/child and child/child.

The *feature/self conflict* occurs when multiple users simultaneously edit the same feature or feature parameters. An example of this is when two users try to edit the same extrude feature and one user inputs 3 mm for the extrude length while the other inputs 5 mm. Since the value for extrude length cannot be both 3 mm and 5 mm, a conflict will occur (see Fig. 2).

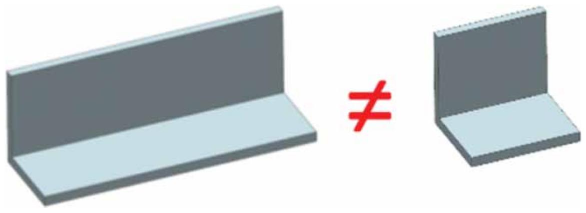


Fig. 2: Feature/self conflict example.

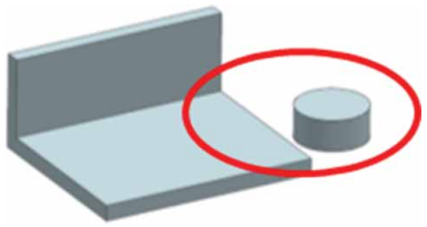


Fig. 3: Parent/child conflict example.

The *parent/child conflict* occurs between a parent feature and a child feature which directly depends on that parent. This happens when a simultaneous creation, edit or deletion of a parent or child causes the other to fail. For example, this occurs if one user creates a hole in an extrusion while a second user changes the extrusion length. Since the hole and the extrusion no longer intersect, this renders the hole invalid (see Fig. 3).

The *child/child conflict* is between two features which reference the same parent geometry. An example of this would be if one user creates a chamfer on an edge while a second user creates a fillet on the same edge. Since these two features reference the same edge and the edge disappears after one of these operations is performed, a conflict results (see Fig. 4). If these various types of syntactic conflicts are not appropriately managed, they will lead to errors within the multi-user CAD system.

In addition to syntactic conflicts, semantic conflicts are also important to manage. These are conflicts which originate from users misunderstanding each other's design intent, which result in operations which cause unintended design results. For example, two designers put a hole on a face at slightly different locations, causing the holes to overlap each other (see Fig. 5). This would not cause a syntactic violation because these are both valid operations that can coexist. However, this causes unintended results in the model and thus violates each of the users' design intent. Semantic conflicts reflect lack of communication between users, not a CAD system architectural flaw. However, managing these conflicts will reduce redundancy and overhead in the parallel design process.

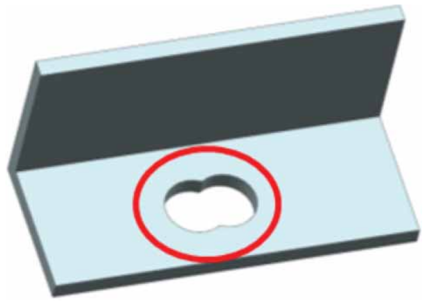


Fig. 5: Semantic conflict example.

In this paper, we focus on the first type of syntactic conflicts, feature/self conflicts, and present a system capable of automatically preventing them. This system also includes methods to communicate feature reservations between users. Visually communicating work areas leads to an overall reduction in semantic conflicts.

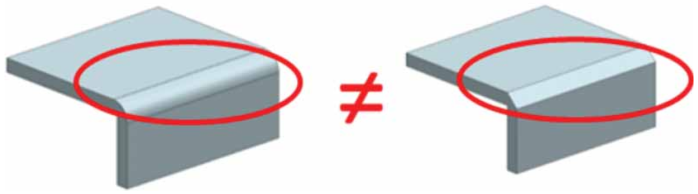


Fig. 4: Child/child conflict example.

3.2. Multi-user CAD Architecture

In order to help the reader understand the conflict management methods discussed herein, we first present the reader with a basic overview of the multi-user CAD system architecture. The system uses a client-server architecture with a thin server and thick client. This requires each user to have a session of the CAD program running on their individual client machines. The system provides each client with identical copies of the CAD part. When an operation occurs, the data to perform that operation is pushed from the source client to the server and subsequently pulled by each of the destination clients. Once the data is pulled, functions are invoked to perform the operation on the destination clients. The server also pushes the operation data to the database where all the data associated with the part is stored persistently. Fig. 6 shows a high level diagram of the software architecture.

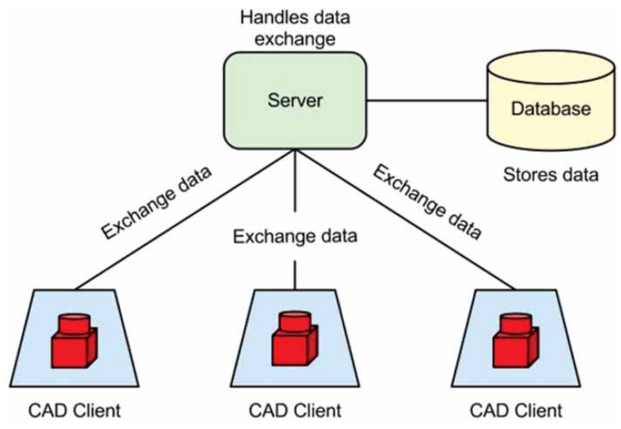


Fig. 6: Thick client multi-user CAD architecture.

3.3. Feature Conflict Avoidance

Commercial CAD systems allow only a single user per part, thus making the part the atomic unit for concurrent CAD interaction with multiple users. With the shift in the CAD modeling paradigm to allow multiple users to modify a part simultaneously, a new atomic unit for multi-user interaction must be established. This research proposes that the atomic unit of a multi-user, feature based CAD system should be the feature. This means that multiple users may not simultaneously modify the same feature within a distributed part. In mathematical terms it can be expressed as follows: Given that F is the set of all features in part P, at a given time interval t, and f is a feature in part P, at time interval t. U is the set of all users in part P, at time interval t and u is a user in part P, at time interval t. The following is true:

$$f \in F(P, t) \tag{3.1}$$

$$u \in U(P, t) \tag{3.2}$$

Axiom 1 states that for a given edit operation there exists a unique set E, in part P, at time interval t, which contains only one feature f and one user u where

$$E(P, t) = \{f, u\} \tag{3.3}$$

For interval

$$t_{beginEdit} \le t \le t_{endEdit} \tag{3.4}$$

Multi-user, feature atomicity is enforced above by only allowing a single instance of set E to exist in part P, at time interval t, where t is the time interval from the beginning to the end of the edit operation. If this atomicity is not enforced, conflicts of a feature with itself will occur when a user on one client edits a feature while another user is editing the same feature.

The process of editing a feature in a CAD system often takes several seconds to perform because multiple parameters for the feature may need to be modified. During the time it takes to complete an edit operation by a user (Laura), a second user (Steve) could potentially make an edit to that same feature. Once Laura finishes the edit, the update is sent to Steve over the server. This update cannot be processed by Steve because he is performing an edit operation and the commercial CAD client can only perform a single operation at a time. Therefore the edit is put into the delayed operation queue. Meanwhile, Laura's update stays consistent on her client with the values she input until Steve finishes his edit. When he finishes, the change is sent over the server to Laura and her feature is modified to the values of Steve's update. After Steve finishes his edit, he will process Laura's update which was sitting in the delayed operation queue. Thus, Laura receives Steve's edit update and Steve receives Laura's edit update. This results in feature data inconsistency between clients after local edits of the same feature (see Fig. 7).

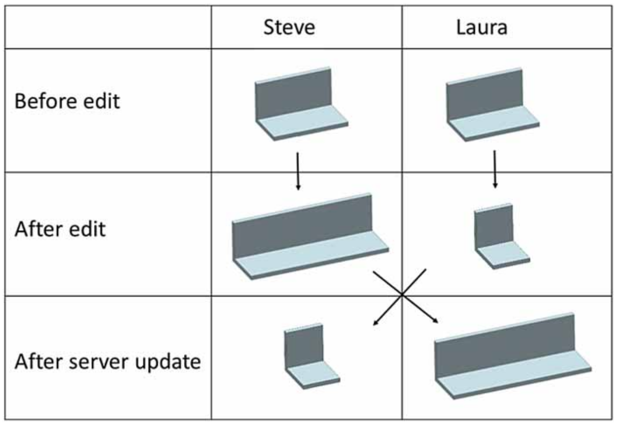


Fig. 7: Simultaneous feature edit leading to feature data inconsistency.

In order to prevent simultaneous edits of a single feature by multiple users, we propose a method for automatic feature reservation and blocking. If a user is editing a feature, this method blocks any other user from editing that feature until the first user finishes his edit. This reservation is communicated by a user sending a message telling the server that the feature is blocked and the server sending a message to all the clients telling them to block the feature. This block makes it so that other users are unable to edit the feature and creates a visual indicator (i.e. color) showing them that it is unavailable. Once the feature edit is complete, a message is sent to the server reporting completion of the edit. The server then sends a message to all clients telling them to remove the feature block. In this way, other users are unable to edit the feature for the duration of a user edit. This is what we call the *client blocking method*.

This method fulfills Axiom 1 if communication between clients and the server is instantaneous. However, due to network latency, this method alone breaks down if multiple users attempt to edit the feature at nearly the same time. Within the time it takes for the client to tell the server to block the feature and for the server to tell all the clients that the feature is blocked, a second client could attempt to make an edit. It is shown that the set E in Eqn. (3.3) is unique only for the interval after the message is received. This accounts for the time it takes to send the block message ($t_{addBlock}$) and the time it takes the send the remove block message ($t_{removeBlock}$):

$$t_{beginEdit} + t_{addBlock} \le t \le t_{endEdit} + t_{removeBlock} \qquad (3.5)$$

Since the interval where set E is unique begins only after the blocking message is received, it is not guaranteed to be unique until after time $t_{addBlock}$. This violates Axiom 1 and results in a potential data conflict scenario as illustrated in Fig. 8. This scenario becomes more likely with higher network latency and as the quantity of concurrent users increases.

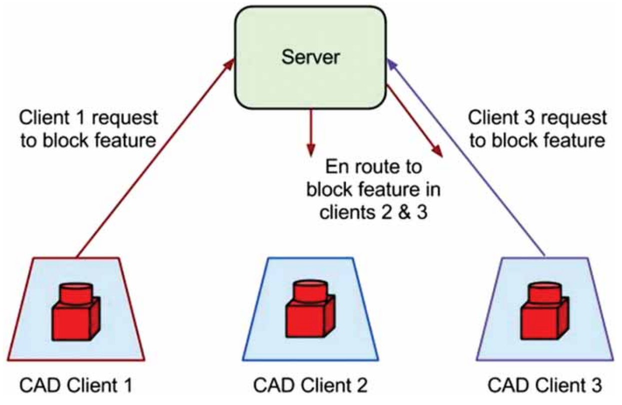

Server
Client 1 request to block feature
En route to block feature in clients 2 & 3
Client 3 request to block feature
CAD Client 1
CAD Client 2
CAD Client 3

Fig. 8: Potential issue with client blocking method.

Additional logic is added to the client blocking method so that multiple simultaneous edits of a feature remain consistent, even if they edit it at approximately the same time. This logic, the *server reservation method*, asynchronously reserves features on the server. An asynchronous approach is used so that a user does not need to wait for a response from the server to begin editing a feature. This method functions as follows: A user attempts to make an edit on a feature that is not yet blocked on the client. A message is sent to the server requesting reservation of that feature. If the feature is not reserved, the server will automatically reserve that feature. This is done by setting a Boolean flag associated with that feature to be true. A message is sent to all other clients telling them that the feature is reserved. All clients will then implement a client block on that feature. No message is sent to the originating client, so he continues to edit the feature assuming he is authorized to make the edit. Once the edit is complete, a message is sent to the server to cancel the feature reservation and set the Boolean flag back to false.

If a client requests reservation on a feature that is already reserved on the server it means that another user is currently editing that feature and the requesting client has not yet received the blocking message. The server will ignore this request. When the requester receives the blocking message he will be ejected from editing the feature (see Fig. 9).

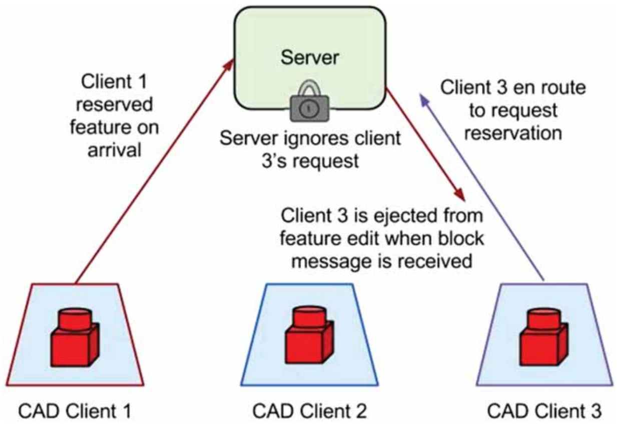

Server
Client 1 reserved feature on arrival
Server ignores client 3's request
Client 3 en route to request reservation
Client 3 is ejected from feature edit when block message is received
CAD Client 1
CAD Client 2
CAD Client 3

Fig. 9: Server reservation method does not allow simultaneous, multi-user feature editing.

The server reservation method provides that the first client to have their request received by the server will be the one authorized to edit a feature. This is because the feature becomes reserved once the message is received, assuming it wasn't reserved previously by another client. Data conflict between a feature and itself does not occur because all other users without the reservation will be rejected from editing the reserved feature. It is shown that since the time interval for set E in eqn. (3.3) exists at the server instead of the clients, the time it takes to send the message is not included in the editing time interval in eqn. (3.4). Assuming that network latency is always less than edit time, Axiom 1 holds true because the edit operation does not truly begin until it reaches and is authorized by the server. Therefore the edit time interval is the time it is reserved on the server and is the same as in eqn. (3.4). Current research is focused on developing methods to maintain consistency even when latency is greater than edit time, however unlikely this case may be.

The combination of the client blocking method with the server reservation method provides the benefits of both approaches. The client blocking method provides that most clients will not even attempt to edit the feature because it is blocked for them both visually and interactively. The server reservation method provides for a safety net in the slight chance that multiple users edit a feature at approximately the same time. These methods provide that only one user is allowed to edit a feature at a time, thus avoiding data consistency problems for a feature with itself.

3.4. On Demand Reservation Removal

Besides the ability to reserve and remove reservation of a feature automatically, it is important to have the ability to remove a reservation on demand if a user is taking an inordinate amount of time editing a feature. For example, if a user is out to lunch while he is in a feature edit, a second user may need to edit that feature and should not have to wait until the first user returns from lunch to do so. One method to do this is to control on demand reservation removal based on user roles. For example, managers or team leads could have the authority to remove reservations. The problem with this approach is that users must find someone with the authority to remove the reservation or else they can't perform the necessary operation. This approach is not very agile because it adds additional steps to the process. Alternately, all users could be allowed to remove reservations at any time. The problem with this approach is that users may actually be in an edit when the reservation is removed and their work would be interrupted.

In order to maximize modeling agility and minimize modeling interruption, the following method is presented for on demand reservation removal. All users are allowed to request reservation removal on any feature that is currently reserved. When a user reservation removal request is initiated, a message is sent to the owner of the feature reservation notifying them that another user would like to remove the feature reservation. The owner has the option to accept or reject the request. If the request is rejected, the reservation remains with the owner and a message is sent to the requester stating that the request is denied. If the request is accepted, the owner is automatically ejected from the edit dialog and the reservation is removed. Alternatively, the owner may choose to ignore the message and continue his edit but is given a limited amount of time to respond to the request. If the owner fails to respond to a request within the allotted time, he is automatically ejected from the edit dialog and the reservation is removed.

This method maximizes modeling agility. All users may request any feature reservation to be removed at any time. Feature reservation removal does not need to involve an authorized user (i.e. manager). On demand reservation removal also minimizes modeling interruption. A user performing an edit is not automatically ejected from their edit. He is notified that a relinquishment request is made, but he retains control of the feature until his time limit to respond expires. Fig. 10 shows a complete flow chart of the combined client blocking, server reservation and on demand reservation removal methods.

4. IMPLEMENTATION IN NXCONNECT

The automated feature conflict avoidance and on demand removal methods have been implemented into a multi-user CAD system being developed at BYU called NXConnect. The development of this software involves utilizing the Siemens NX application programming interface (API) to extend the current single user functionality of NX to be a simultaneous multi-user application. NXConnect uses a client-server architecture with a thin server and thick client which requires an instance of an NX session running on each client. Each client that is participating in a given model has an identical copy of the NX part that stays in sync with all other clients in that same model. Data consistency between clients is achieved by propagating model changes to the server and then onto other clients. When clients receive data pertaining to an operation on another client, the NX API function to perform that operation is called on the clients' session so the clients all get changes performed by other users. The data to perform that operation, as well as any geometry reference data, is recorded in a database for future retrieval [13,15,16,20].

The feature reservation and blocking methods have been implemented in NXConnect as follows: the server has an up-to-date reservation list which contains all the reservations held by users in each part. When a user attempts to edit a feature or features, an event for that feature is activated, which triggers a message to the server requesting access to edit that feature. The server handles the message by checking the reservation list to see whether the specific feature has already been reserved by another user. If the feature is not reserved, the reservation is added to the reservation list to prevent other users from editing the feature. Access is implicitly granted to the requester by not sending a message back. Since the reservation messages are asynchronous, a user does not need to wait for a message back from the server to begin editing a feature. Therefore, if he does not receive a rejection message, he is safe to continue editing. However, if the feature is already

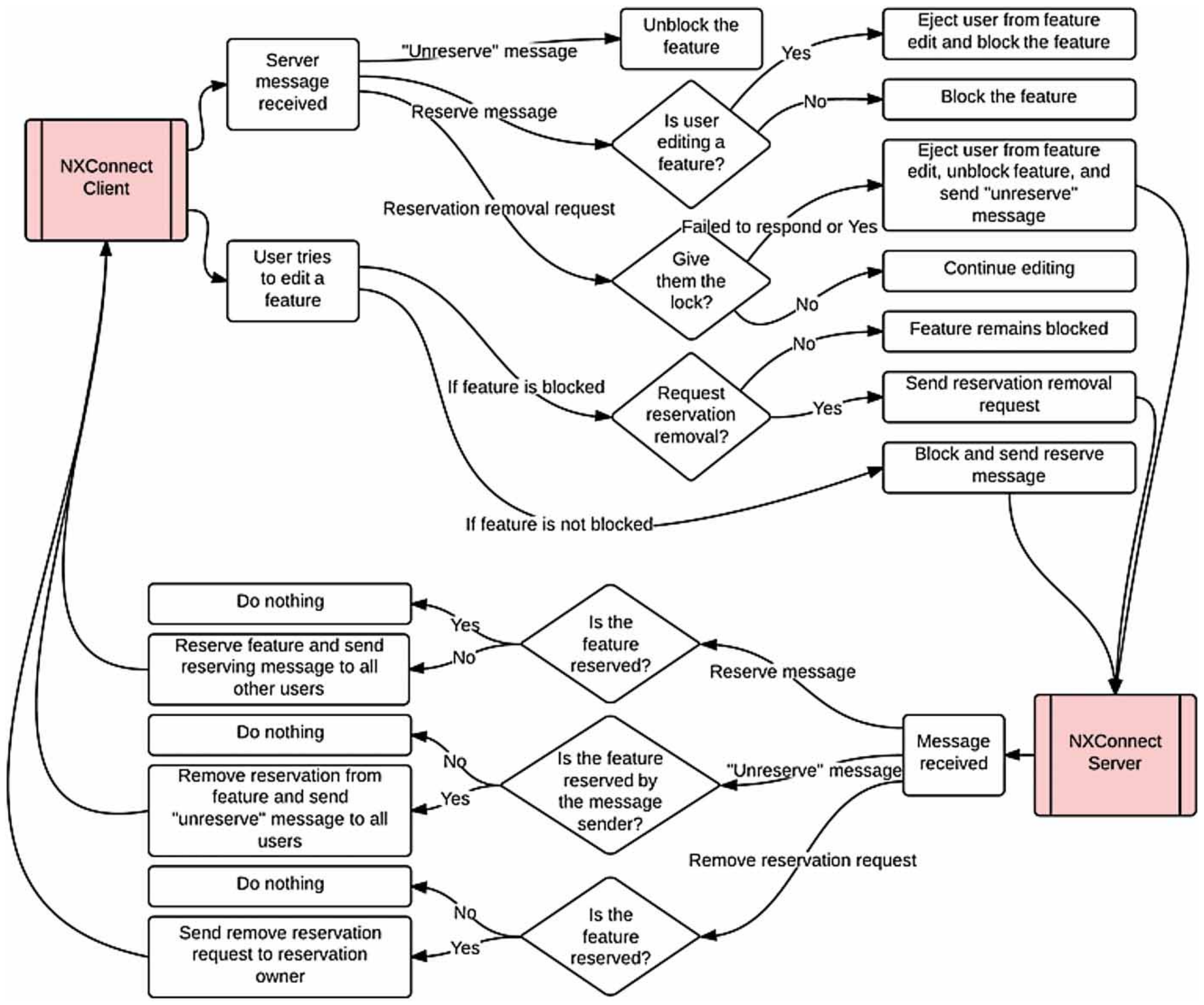
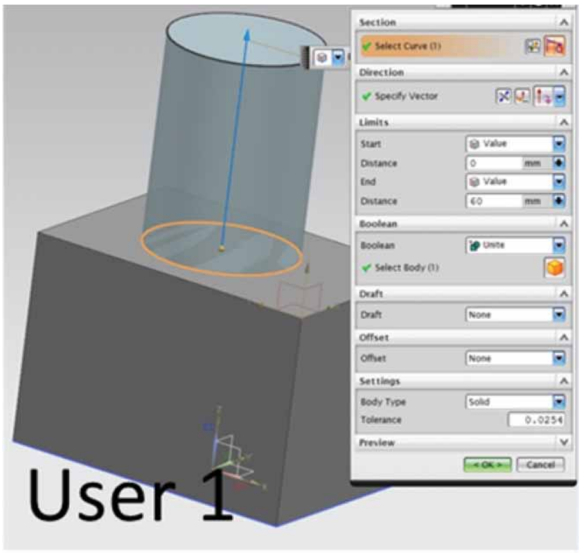

NXConnect Client
Server message received
"Unreserve" message
Unblock the feature
Reserve message
Is user editing a feature?
Yes
Eject user from feature edit and block the feature
No
Block the feature
Reservation removal request
Give them the lock?
Failed to respond or Yes
Eject user from feature edit, unblock feature, and send "unreserve" message
No
Continue editing
User tries to edit a feature
If feature is blocked
Request reservation removal?
No
Feature remains blocked
Yes
Send reservation removal request
If feature is not blocked
Block and send reserve message
Do nothing
Yes
No
Reserve feature and send reserving message to all other users
Is the feature reserved?
Reserve message
Message received
NXConnect Server
Do nothing
No
Yes
Remove reservation from feature and send "unreserve" message to all users
Is the feature reserved by the message sender?
"Unreserve" message
Remove reservation request
Do nothing
No
Yes
Send remove reservation request to reservation owner
Is the feature reserved?

Fig. 10: Flow chart of combined client blocking, server reservation and on demand reservation removal methods.

reserved, a message is sent to the requester indicating the feature is reserved by another user. When this message is received, the edit dialog is automatically exited to prevent the user from editing the feature. Therefore feature reservation is handled even when multiple users request a reservation at nearly the same time. This is implemented in such a way that reservation can be made for multiple feature edits as well.

When access is granted to edit a feature, a message is sent to all other clients blocking that particular NX feature. When a feature is blocked on the client, it is colored with a color users are trained to recognize as blocked. If the blocked feature is a solid body, all the faces of that body are colored. If the blocked feature is a sketch or curve, the curves are assigned the blocking color. All other features and curves are left to the default color. This communicates that the feature is reserved and not available for editing, while all other features (of the default color) are still available. Figures 11 and 12 show the NXConnect reservation implementation: the circle extrusion feature is edited on User 1's session and is reserved for editing in User 2's session (shown in red in Fig. 12).

In addition to coloring the feature, the feature is blocked on the client to prevent users from editing the reserved feature. A client-side status flag, associated with each feature on a given client, keeps track of whether a feature is "reserved by another user" (RBA), "reserved by me" (RBM) or "not reserved" (NR). The default is set to NR. When the user reserves a feature, the status is set to RBM. It is important to know that a given feature is reserved by the user because when he releases a reservation after an edit, he needs to know which features were reserved in order to remove the reservation. When a reservation message is received from the server, the status flag is set to RBA. Whenever a user attempts to edit a feature, the client first checks to see whether the client-side feature status flag is set to RBA. If this is the case, the edit operation will automatically be canceled, thus preventing users from editing a feature reserved by another user.



Start: Value
Distance: 0 mm
End: Value
Distance: 60 mm
Boolean: Unite
Draft: None
Offset: None
Body Type: Solid
Tolerance: 0.0254

Fig. 11: User 1 editing extrusion of circle.

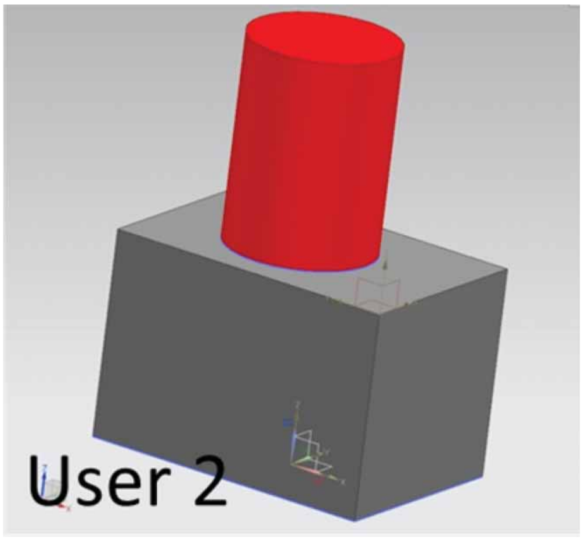
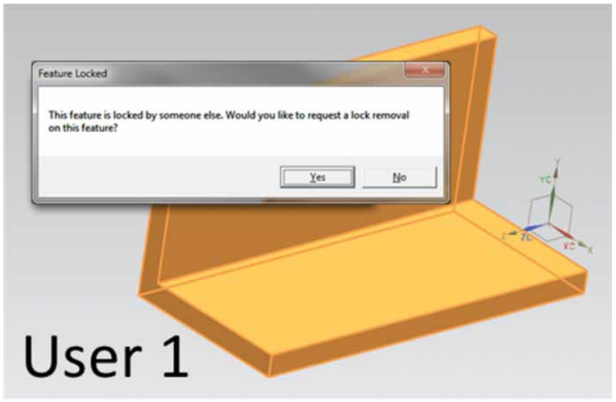


Fig. 12: User 2 has the circle extrusion reserved (red).

Since feature reservations only remain for the duration of the feature edit operation, reservations are removed automatically once the operation is complete. When a user finishes editing a feature, a message is sent to the server requesting the reservation be removed for all features which are set to RBM. When the server receives this message it will remove the reserved feature from the reservation list and send a message to all clients telling them to remove the client block. The clients each handle the message by changing the reservation status to NR and changing the color of the feature back to the default color. In this way the feature becomes available to edit and the client block is removed.

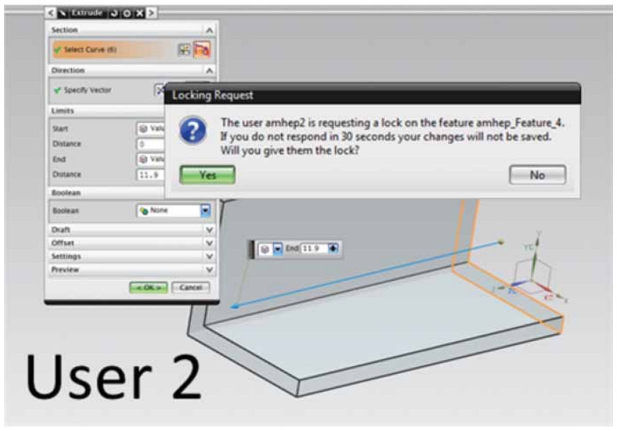
When a user opens a part in which users already have features reserved, it is essential that the client receive all existing reservations in the part. Additionally, when a user logs out or exits a part, all the reservations they have in place must be removed. This is done as follows: when a user loads a given part, the reservation list is queried for all feature reservations in that part. Reservation messages are sent to the user for each feature reservation in the part and the features are blocked according to the client blocking method. When a user logs out or exits a part, messages are sent requesting removal of all reservations for that client. In this way reservations are added and removed by clients who join or exit a modeling session.

On demand feature reservation removal is implemented in NXConnect as well. A user requests reservation removal by double clicking on a feature that is blocked on the client. He receives a message notifying him that the feature is reserved and asks him if he would like to request removal of the reservation. If he clicks yes, a request is sent to the owner via the message server. The reservation owner receives the request message and has 30 seconds to respond, during which time the user may continue to edit the feature. He may also choose to reject the request which allows him to continue editing the feature until another request is received. It also sends a message to the requester notifying him that the request is denied. Conversely, if the reservation owner accepts the removal request, he is automatically ejected from the edit dialog and the reservation is removed. Additionally, if he fails to respond to the message within the 30 second period, he is ejected from the edit dialog and the reservation is removed. Fig. 13 shows the message received when User 1 tries to edit a feature. He decides to request a reservation removal and User 2, who owns the reservation, receives the message as seen in Fig. 14. This message explains that User 2 can either accept or reject the reservation removal request and that he must respond in 30 seconds; otherwise the reservation is automatically revoked and he is ejected from the feature edit.



This feature is locked by someone else. Would you like to request a lock removal on this feature?

Fig. 13: User 1 requests a reservation removal.


Locking Request

The user amhep2 is requesting a lock on the feature amhep_Feature_4. If you do not respond in 30 seconds your changes will not be saved. Will you give them the lock?

Fig. 14: User 2 receives a reservation removal request.

5. RESULTS

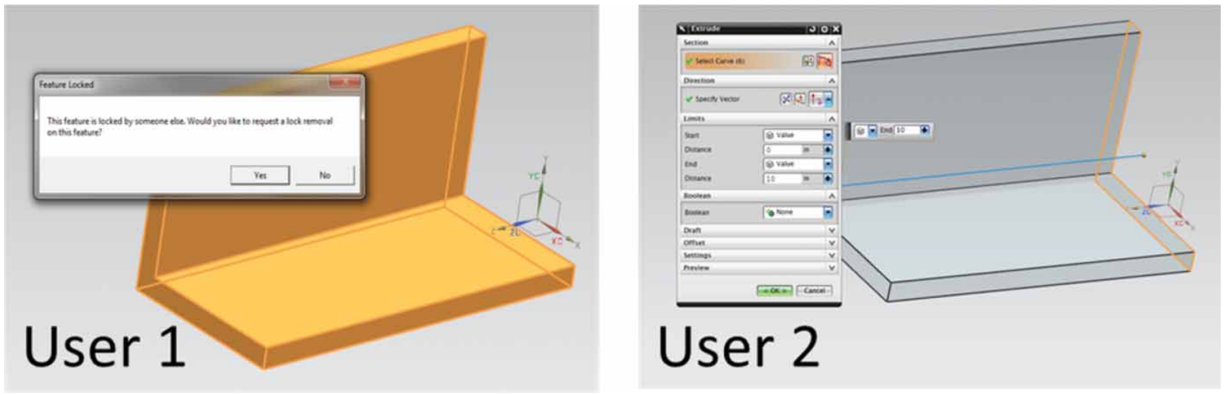
To show the validity of the automated feature reservation method for preventing *feature/self conflicts*, two test cases were performed in NXConnect. Both of these tests are performed on a simple angle iron model where two users try to edit the extrusion length at the same time. To simulate simultaneous edits, two users purposely click on the feature edit command at the same time. The first case is performed with the feature reservation methods in place and primarily tests the implementation of the client blocking method. The second case is performed with an added 2.5 second latency on the network to provide enough latency to test the implementation of the server reservation method.

The first test case has two users try to edit the extrusion length at the same time. The result of this test case is that the first user (User 2) to click on the edit feature operation is able to perform the edit. The second user (User 1) is not allowed to edit the feature because the block is already in place by the time the user clicks (see Fig. 15). This is due to the fact that network latency is typically less than human reaction time. The chances that two users will send reservation requests within typical message round trip time (RTT) is very small. However, the second test is in place to verify that if this happened, the feature reservation system still remains robust.

The second test case has two users try to edit the extrusion length at the same time, but has an added 2.5 second latency on the network. This provides sufficient time for both users to send a request message to the server before one receives the reserve message. The result of the second test case is that the first user to click on the edit feature operation is able to perform the edit (User 1 in Fig. 16). The second user seems to be allowed to edit the feature but is ejected from the feature edit in less than 2.5 seconds (User 2 in Fig. 16).

The upper two images in Fig. 16 shows two users being able to simultaneously edit the extrusion for a limited time. However, shortly after, User 2 is ejected from the edit dialog and the feature is blocked on his client (see lower two images in Fig. 16). This happens because User 1's reservation request arrived to the server first. When User 2's request was received by the server, it is rejected because the feature was already reserved by User 1. Once he receives a message from the server rejecting his request, he is automatically ejected from the dialog and the feature turns red signifying a feature block. Assuming network latency is always less than edit time, the user whose request is rejected will never be able to complete the edit before he is ejected from the edit dialog.

The results from these two tests show that this methodology prevents the data inconsistency that results from feature/self conflicts as seen in Fig. 7. Latency is not only the limiting factor in performance of the system, but it tends to create many cases that can cause data inconsistency between two clients. Maintaining data consistency for a distributed system in a wide-area network is challenging, but this system lays the groundwork for doing so with many users and typical latency [9].


Feature Locked

This feature is locked by someone else. Would you like to request a lock removal on this feature?

Fig. 15: Both users tried to edit the feature at the same time and User 2 get to the server first so User 1 receives the block notification from the server.

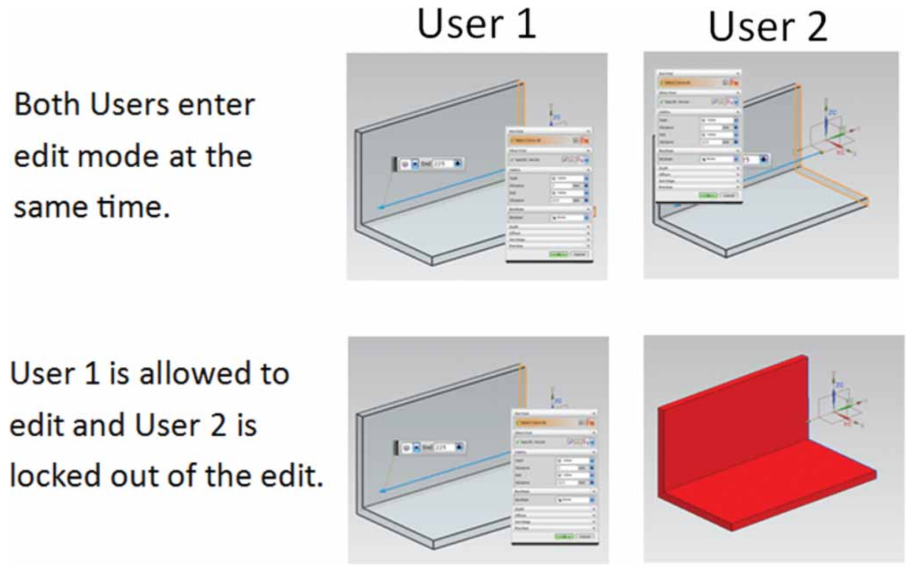

User 1
User 2
Both Users enter edit mode at the same time.
User 1 is allowed to edit and User 2 is locked out of the edit.

Fig. 16: Both users are allowed in the edit until one has received a rejection message and then blocked from the feature edit.

6. CONCLUSIONS

An automated feature conflict avoidance methodology is presented which prevents multiple users from simultaneously editing the same feature. This approach automatically reserves a feature for the duration of an edit with priority given to the first client to request the reservation. A method is also presented for agile on demand reservation removal. Methods are presented that create an asynchronous feature reservation on the server and a block on the client. Client blocks have a visual cue to communicate the reservation status and forcibly prevent users from editing a reserved feature. Server reservations are in place to prevent feature conflicts when multiple users edit the same feature at nearly the same time. It is shown how this method fulfills Axiom 1 to preserve the atomicity of a single user and feature pair for the duration of a feature edit, assuming that network latency is less than edit time.

This approach shows the ability and functionality of automated feature reservation to prevent conflicts, thus helping to enable an agile collaborative environment for concurrent engineering. This approach has less overhead than other approaches because users do not need to manually manage or merge conflicts, as is the case with other approaches. The user is not required to intelligently manage conflicts on their own (potentially allowing inconsistencies); rather, the CAD system handles them automatically with limited user interruption. Reservation information is communicated to users via non-intrusive visual cues. Additionally, users are not constrained by artificial boundaries that may prevent them from contributing where they are needed. This approach preserves the agility of a truly parallel design workflow while still eliminating feature/self conflicts. Current research at the NSF Center for e-Design, BYU site is focused on developing methods to maintain consistency for any latency and applying rules-based methods to manage parent/child and child/child conflicts.

ACKNOWLEDGEMENTS

Special thanks to The Boeing Company for funding this research.

REFERENCES

[1] Agustina, A.; F. Liu; S. Xia; H. Shen; C. Sun: CoMaya: incorporating advanced collaboration capabilities into 3d digital media design tools, Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work, 2008, 5–8.

[2] Bidarra, R.; E. van den Berg; W. F. Bronsvoort: A Collaborative Feature Modeling System, Journal of Computing and Information Science in Engineering, 2(3), 2002, 192, http://dx.doi.org/10.1115/1.1521435.

[3] Bu, J.; B. Jiang; C. Chen: Maintaining semantic consistency in real-time collaborative graphics editing systems, IJCSNS, 6(4), 2006, 57.

[4] Cannon, L.; Nysetvold, T.; Phelps, G.; Winn, J.; Jensen C. G.: How Can NX Advanced Simulation Support Multi-User Design?, Computer-Aided Design and Applications, PACE Vol. 2, 2012, 21–32.

[5] Cera, C.; I. Braude; I. Comer; T. Kim; J. Han; W. Regli: Hierarchical Role-Based Viewing for Secure Collaborative CAD, in Proceedings of the 2003 ASME International Design Engineering Technical Conferences & The Computer

and Information in Engineering Conference (DETC/CIE2003), 2003, 10.

[6] Chan, S.; M. Wong; V. Ng: Collaborative solid modeling on the WWW, Proceedings of the 1999 ACM symposium on Applied computing - SAC '99, 1999, 598–602, 10.1145/298151.298487.

[7] Chen, L; Song, Zhijie; Feng, L.: Internet-enabled real-time collaborative assembly modeling via an e-Assembly system: status and promise, Computer-Aided Design, 36(9), 2004, 835–847, http://dx.doi.org/10.1016/j.cad.2003.09.010.

[8] Ellis, C.A.; Gibbs, S.J.: Concurrency control in groupware systems. ACM SIGMOD Record 18 (2), 1989, 399–407

[9] Federal Communication Commission: A Report on Consumer Wireline Broadband Performance in the U.S, http://www.fcc.gov/measuring-broadband-america/2012/july#Findings. 2012

[10] Jing, S.; F. He; S. Han; X. Cai; H. J. Liu: A method for topological entity correspondence in a replicated collaborative CAD system, Computers in Industry, 60(7), 2009, 467–475, http://dx.doi.org/10.1016/j.compind.2009.02.005.

[11] Li, W. D.; J. Y. H. Fuh; Y. S. Wong: An Internet- enabled integrated system for co-design and concurrent engineering, Computers in Industry, 55(1), 2004, 87–103, http://dx.doi.org/10.1016/j.compind.2003.10.010.

[12] Lin, K.; D. Chen; C. Sun; G. Dromey: Maintaining constraints in collaborative graphic systems: the CoGSE approach, in ECSCW 2005, (September), 2005, 185–204.

[13] Marshall, F.: Model Decomposition and Constraints to Parametrically Partition Design Space in a Collaborative CAx Environment, Brigham Young University, 2011.

[14] Moncur, R.; Jensen, C.; Teng, C.; Red, E.: Data Consistency and Conflict Avoidance in a Multi-User CAx Environment, 10(5), 2013, 727–744.

[15] Red, E.; Jensen, C.; Holyoak, V.; Marshall, F.; Xu, Y.: v-Cax: A Research Agenda for Collaborative Computer-Aided Applications, 7(3), 2010, 387–404.

[16] Red, E.; Jensen, C.; French, D.; Weerakoon, P.: Multi-User Architectures for Computer-Aided Engineering Collaboration, International Conference on Concurrent Enterprising, 2011.

[17] Saito, Y.; Shapiro, M.: Optimistic Replication, ACM Computing Surveys, 37 (1), 2005, 42–81.

[18] Shen, L; Hao, Y; Li, M; Zhao, W; Zheng, J.: A Synchronous Collaborative Environment for Engineering Design Education, International Conference on Computer Supported Cooperative Work in Design, 11th, 2007: 298–303.

[19] Sun, C.; Xia, S.; Sun, D.; Chen, D.; Shen, H.; Cai, W.: Transparent Adaptation of Single-User Applications for Multi-User Real-Time Collaboration, ACM Transactions on Computer-Human Interaction, Vol. 13, No. 4, December 2006, 531–582.

[20] Xu, Y; Edward Red, E.; Jensen, C.: A Flexible Context Architecture for a Multi-User GUI, Computer-Aided Design & Applications, 8(4), 2011, 479–497.