Data Consistency and Conflict Avoidance in a Multi-User CAx Environment

Robert Aaron Moncur
Brigham Young University - Provo

Data Consistency and Conflict Avoidance in a Multi-User CAx Environment

Robert A. Moncur

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

July 2012

ABSTRACT

Data Consistency and Conflict Avoidance in a Multi-User CAx Environment

Robert A. Moncur
Department of Mechanical Engineering, BYU
Master of Science

This research presents a new method to preserve data consistency in a multi-user CAx environment. The new method includes three types of constraints which work by constraining and controlling both features and users across an entire multi-user CAx platform. The first type of constraint includes locking or reserving features to enable only one user at a time to edit a given feature. The second type of constraint, collaborative feature constraints, allows flexible constraining of each individual feature in a model, and the data that defines it. The third type of constraint, collaborative user constraints, allows the constraining of user permissions and user actions individually or as a group while providing as much flexibility as possible.

To further present this method, mock-ups and suggested implementation guidelines are presented. To demonstrate the effectiveness of the method, a proof-of-concept implementation was built using the CATIA Connect multi-user CAD prototype developed at BYU. Using this implementation usage examples are provided to show how this method provides important tools that increase collaborative capabilities to a multi-user CAx system. By using the suggested method design teams will be able to better control how their data is used and edited, maintaining better data consistency and preventing data conflict and data misuse.

Keywords: Multi-user CAD, data consistency, collaborative design

ACKNOWLEDGEMENTS

I would like to thank my family (especially my darling wife-to-be Lisa) for their support and encouragement as I completed this work. I would also like to thank my advisor Dr. Jensen and committee members Dr. Teng, and Dr. Red for their input and guidance in completing this research. Additionally I would like to thank the NSF for their funding and support of the Center for e-Design, BYU site in which this thesis and research were completed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Engineering and product development, by nature, is a highly collaborative activity. With increasing globalization of the world economy and the pressure for companies to shorten product development time, the need for better collaborative engineering tools has arisen. With the widespread reach of the internet, as well as the ever-increasing network bandwidths and computer processing speeds, the technologies needed to improve collaboration in modern CAx systems are in place. Unfortunately, single-user serial architectures in modern commercial CAx systems inhibit concurrent engineering, in spite of the numerous research efforts into product team cooperation, functional constraints, and data/model propagation and transparency [1]. To improve the collaborative capacity of modern CAx systems, further research and development is necessary.

1.1 Problem Statement

Traditionally, PLM applications manage data conflicts by restricting file editing access to a single user, using secured file check-out and check-in methods. Multi-user tools will require simultaneous editing access to those same files by a collaborating group, with new requirements for data flow management and data consistency in a distributed user network. Without an effective method to manage CAx data consistency (maintaining the desired form and intent for the data in the midst of several users working on the same model and possibly the same features within that model) multi-user CAx tools become useless as would any software tool that did not maintain the integrity of a user's data.

1.2 Thesis Objective

The primary research objective is to develop a data consistency architecture and system to protect data and prevent data conflict in a collaborative CAx environment. First, situations that could compromise data consistency and/or introduce data conflict in a collaborative design session will be introduced. Next a methodology and implementation for avoiding and/or preventing these situations will be presented. This system will include a user interface, underlying data-model, and procedure that could be adapted to an existing CAx system through its API. The user-interface, data-model, and procedure for the system will then be created and implemented into the multi-user CAD plugin CATIA Connect although it could be developed using a similar collaborative system working with one of several commercial CAx systems, as the basic data structure of these systems would be adaptable to such a system. Finally usage examples will be given, methods for comparing this method to other the data consistency systems will be discussed, and conclusions will be given.

1.3 Problem Delimitation

It is important to note that this research is limited to demonstrating only that the developed data consistency method improves collaboration. The software tools created for this research are prototypes meant to be a proof-of-concept of the method described in this thesis, and are not meant for distribution and thus lack the overall depth and sophistication of enterprise software. While this new system provides an improved collaboration experience, it may not be the best possible implementation of the tool, but is rather an embodiment of the proposed method.

The method implementation will only be created for and tested using CATIA Connect CAD running on a Microsoft Windows operating system with version 4.0 of the .NET platform. Within that CAD package only a restricted subset of the CAD feature functionality has been implemented to interface with the method described in this thesis. The scope of this research will be general enough to apply to all CAE modeling operations, but the implementation will be limited to the basic CAD modeling operations of positive and negative extrusions formed with sketches of simple line and circle geometry. Additionally it is to be noted that current exposed API functionality of the given CAD system may introduce additional limitations in the implementation that could otherwise be developed given additional API functionality. Finally, it should be noted that these delimitations are not the result of limitations on the capabilities of the proposed method and that future work could extend these implementations to produce a production quality implementation.

1.4 Document Organization

The next chapter will review relative prior research in the fields of collaborative CAD and data consistency management. Chapter three presents the generalized method for improving collaboration through the use of a data consistency management system. Chapter four describes the implementation of this method using the API of the previously mentioned CAD system. Chapter five presents a detailed usage example showing how the method could be used in a collaborative design session, the results of functional observations, and a flexible method to compared different data consistency management systems. Finally, chapter six details conclusions that have been found from this research and presents suggestions for future work.

2 BACKGROUND

The method presented in this paper builds on the work done by other researchers. This section has been divided into four relevant sections: Collaboration in CAx Software, About Data Conflict, Current Constraint and Conflict Resolution Technology, and Current Data Consistency Methods in CAx.

2.1 Collaboration and CAx

Over the last ten years researchers have implemented collaborative functionality into CAx tools. This development has been done in two ways; the first being several new CAx tools have been developed from the ground up, creating the software architecture in such a way that collaborative features would be native to the software. Some of these software packages include web-based systems such as WebSPIFF [2], WebCOSMOS [3], CADDAC [4], and NetFeature [5]. In these systems there is a central modeling server where all modeling operations are executed and the central CAD model stored. The clients are responsible for visualization tasks and interaction between the clients and server. The data transmitted on the network is the faceted CAD model data, which leads to heavy network traffic load. Approaches such as incremental faceted model transmission adopted in WebCOSMOS [3], macro-files adopted in WPDSS [6] are proposed to alleviate the data transmission problem, but the transmission problem still exists.

Additional collaborative CAx systems include CollabCAD and NXCollaborate. Most of these systems were merely prototypes and never developed into commercial-grade software.

The second method that has been used to develop collaborative CAx applications is called transparent adaptation, which simply means that plugins that provide collaborative capabilities are developed for existing CAx systems. This method allows the simplification of adding of collaborative capabilities into already popular and industry accepted tools. Transparent adaptation tools are implemented through the API of existing systems and thus no code changes are needed to be made to existing commercial-grade software code to implement collaborative features. However, this approach is limited by the exposed functionality of the given CAx system's API, which in some software packages is not robust enough to effectively implement to support a collaborative plugin. Examples of transparent adaptation implementations include NXConnect [7], a plugin for Siemens NX 6.0 that synchronizes CAD geometry between multiple users, and CoMaya [8], a plugin for Autodesk Maya that also synchronizes animation geometry between multiple users. Additional examples of transparent adaptation collaborative plugins for the Microsoft office software are developed by CodoxWare [9].

2.2 About Data Conflict

A collaborative CAx system based on a relational database is particularly vulnerable to *update conflicts* and *delete conflicts*. This vulnerability arises as several users are updating and deleting the same data simultaneously in the CAx system. An update conflict occurs when the replication of an update to a row (of data in a database) conflicts with another update to the same row. Update conflicts can happen when two database transactions originating from different sites update the same row at nearly the same time. An update conflict could also occur if one client has several local modifications which have not been propagated to the database (possibly due to an interrupted network connection) then those changes are all propagated at once when the network connection comes back online. If there are any modifications to existing data that happened while the network connection was interrupted, an update conflict could occur. A delete conflict occurs when two transactions originate from different sites, with one transaction deleting a row and another transaction updating or deleting the same row, because in this case the row does not exist to be either updated or deleted [10].

2.3 Data Consistency Approaches

Two major approaches have been used by researchers in constraining and controlling the flow of data in collaborative software. The first approach, the *optimistic* approach, is an approach where no restriction or locking of the data takes place in the system, thus resulting in a less restricted feel by users, but also creating the potential for more challenges in data consistency management. Such an approach has been effectively used in multi-user document editors such as Google Docs, CodoxWord, and Microsoft SharePoint. To the author's knowledge a truly optimistic approach has never been successfully implemented in a CAx system, likely due to the complexity of the data relationships within a CAx system. Augustina et al. explain how such an approach is not possible using traditional collaborative data transfer technology such as operational transformations because many data objects could have both multiple parents and multiple children, thus not allowing the use of traditional operational transform technology to manage data as is done in the multi-use text editors [8].

The second—and most common—approach for data consistency control in multi-user CAx systems is the *realistic* (sometimes knows as *pessimistic*) approach, which involves locking parts of the entire CAx model to prevent problems with data consistency. Advantages of realistic approaches to data consistency in CAx systems are easy implementation (as no special transformations of data are required as data is transferred) and the effective prevention of both update and delete conflicts. The strictest realistic approaches fully lock the collaborative design session allowing only one user to edit the model at a time, and the remaining collaborative users to have a view-only mode of the model. More segmented approaches have been implemented in some of the Microsoft Office 2010 collaborative document editing software, allowing certain segments of the document to be locked for editing, while allowing the rest of the document to be open for editing from other collaborators. Changes to the document are not propagated automatically, rather are broadcast to other users when the document is saved and when the user chooses to broadcast the changes.

2.4 Data Consistency Implementations

Several methods for managing data consistency in a collaborative setting have already been proposed in current research. Jing, et al. use a local locking mechanism, based on operational transforms, which helps avoid conflicts by distributing locks to local features in a model in a replicated collaborative CAx system [11]. Bu, et al. use a semantic locking method to prevent semantic violation. The locks are classified into region lock and object lock for resolving violations at different levels of detail. User negotiation and versioning rules are used to resolve conflicts among the collaborating users [12]. Chen, et al. apply three coordination rules embedded in e-Assembly to satisfy collaborative assembly constraints in a client-server environment. The coordination rules serve to maintain consistency among collaborators working on different aspects of an assembly (atomic object, object links, and object interface constraints) [13]. Lin, et al. explains the difficulty of applying constraint methods when several collaborators are concurrently engaged in graphic design, while noting that constraint methods are proven tools in single-user systems. Lin considers collaborative locking, masking, and time stamped methods when constraints are applied to design operations such as moving a graphical object that is subject to some geometrical or parametric constraint, and where multiple users manipulating the same object would confuse the constraint relations [14]. Marshall recommends a task-based method that will allow hierarchal administrative control over nearly the entire design process. This system is very intricate and requires significant planning and setup to be effective [15]. Cera, et al. recommends a role-based system that only provides necessary geometry of a collaborative CAD model based on a user's role. Additionally details of the model can be obscured to protect proprietary information in the model [16]. A more realistic method proposed uses a token-based session, which only allows one user to edit the part at a time [17][18]. Additionally a softer "traffic light" method which visually warns users when it is ok to edit model features was proposed by Bidarra et al. to lift some strict restrictions imposed by the more rigid token-based systems [2].

3 METHODS

To facilitate CAx users to be able to comfortably work with a collaborative system (and ease the fear of losing work due to data problems), a new method to protect and coordinate the transfer of data as well as avoid data conflict and data loss has been developed.

This section presents the general method for the improvement of data consistency in a collaborative CAx system. Note that while the method described is largely in the context of a CAD system the method described is intended to be universally applicable to any collaborative CAx system. For example, in the context of a collaborative CAD assembly, rather than CAD part modeling, each part in the assembly could be treated as a feature and this method could be an effective data consistency system for a CAD assembly system as well. The method steps will be demonstrated with many examples and references in the context of a CAD system; however, for the full details of the implementation please refer to section four.

3.1 Usability and Design

For a collaborative CAx data consistency method to be effective, it must appeal to the user both visually and functionally. Such a system needs to be both simple enough to easily understand, while versatile enough to provide useful and flexible functionality. Applying modern software usability and design principles, this method has considered the end-user to provide an intuitive and usable method to control data in a collaborative CAx setting. It is designed to give the user instantaneous visual feedback about the flow of CAx data in a collaborative system, and when it is and is not safe or allowed for them to edit existing features, delete unwanted features, or add additional features. Additionally the system was designed to adapt to both very strictly controlled and very open design collaborations to be able to work effectively in both contexts.

3.2 System Overview

This section gives an overview of the method.

Figure 3-1 represents the data consistency system for this method running on one client (of possible several connected to the database) in a multi-user CAx design session and shows how the multi-user CAx layer works with the database to synchronize model data. This chapter will explain how this data consistency system functions to protect CAx data in a multi-user environment.

3.3 Data Consistency User Interface

First, the "front end" user-interactive parts of this method will be explained. This will explain both the collaborative design session interface as well as an administrative form to help manage the data consistency system as well as create collaborative user constraints.

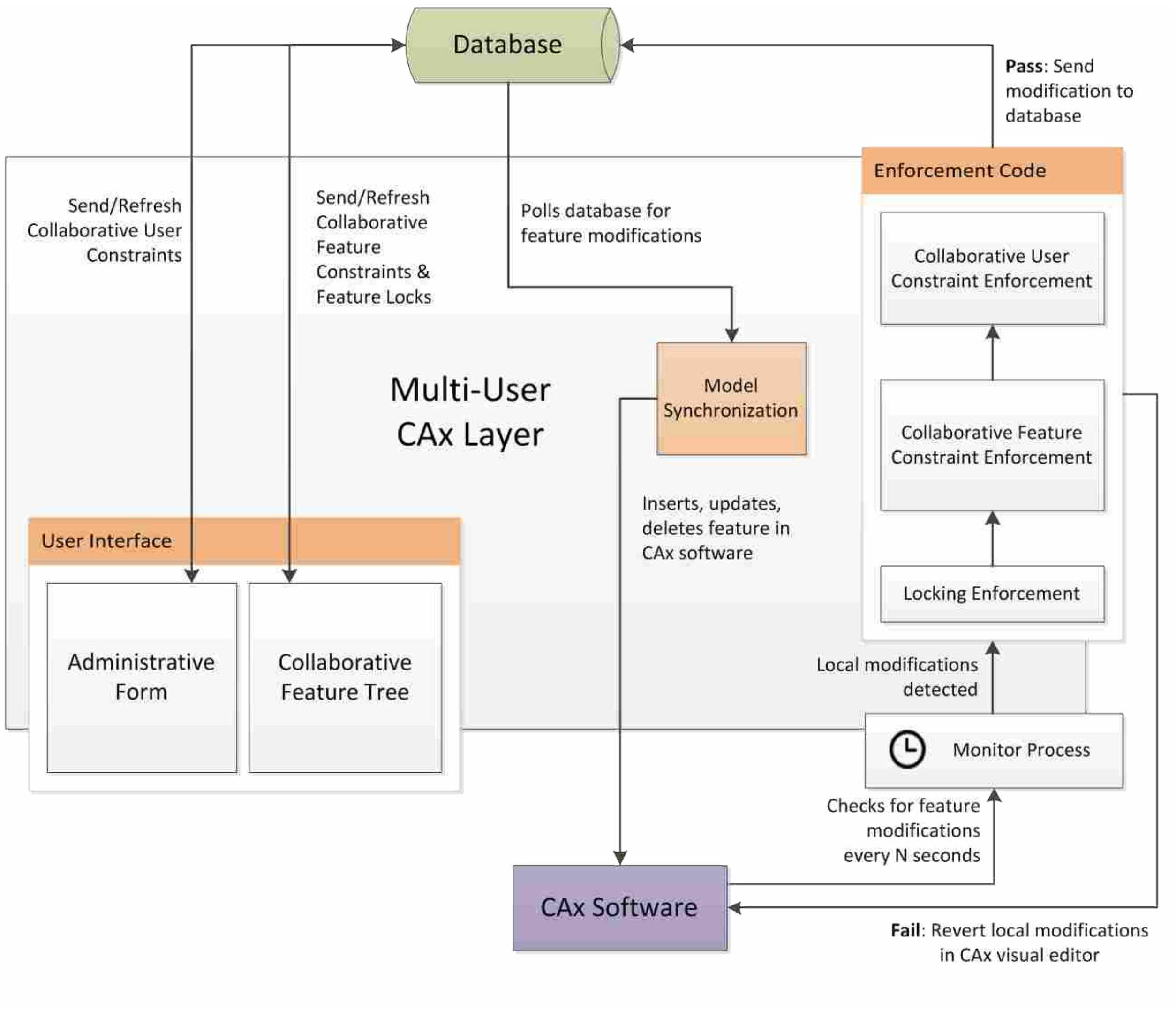
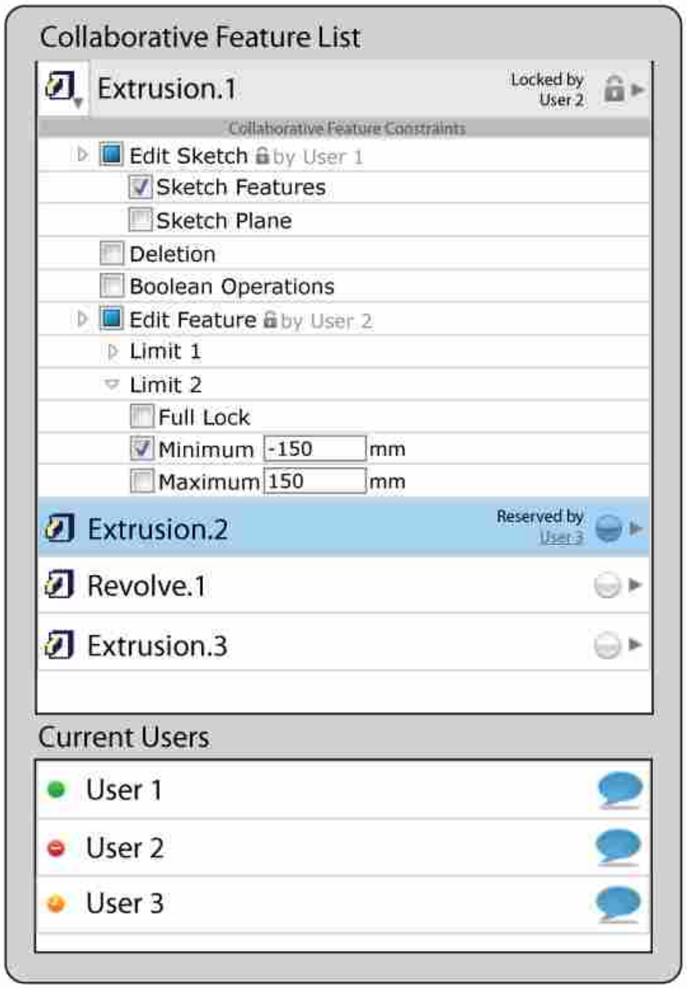

Database
Pass: Send modification to database
Enforcement Code
Send/Refresh Collaborative User Constraints
Send/Refresh Collaborative Feature Constraints & Feature Locks
Polls database for feature modifications
Collaborative User Constraint Enforcement
Model Synchronization
Multi-User CAx Layer
Collaborative Feature Constraint Enforcement
Inserts, updates, deletes feature in CAx software
User Interface
Locking Enforcement
Administrative Form
Collaborative Feature Tree
Local modifications detected
Monitor Process
Checks for feature modifications every N seconds
CAx Software
Fail: Revert local modifications in CAx visual editor

Figure 3-1: Data consistency system overview

3.3.1 Feature List & Feature Reservation

This method for maintaining data consistency is controlled by CAx software users through an information window that displays a representation of a part's feature list. From this form there will be several options that users in a collaborative design session can access to control when and how the different features in a model can be edited. The appearance of this collaborative feature list form could vary depending on the implementation of the collaborative system (fully integrated vs. transparent adaptation) and the style/design of the software that it is being implemented into, but the basic functionality should remain the same. Figure 3-2 is a mock-up of a collaborative feature list (see Figure 4-2 for the implementation of this form).



Collaborative Feature List

Extrusion.1 — Locked by User 2

Collaborative Feature Constraints

- Edit Sketch by User 1
  - [✓] Sketch Features
  - [ ] Sketch Plane
- [ ] Deletion
- [ ] Boolean Operations
- Edit Feature by User 2
  - Limit 1
  - Limit 2
    - [ ] Full Lock
    - [✓] Minimum -150 mm
    - [ ] Maximum 150 mm

Extrusion.2 — Reserved by User 3

Revolve.1

Extrusion.3

Current Users

- User 1
- User 2
- User 3

Figure 3-2: Mock-up of a collaborative feature list form

The most basic functionality of the feature list *reserves* or *locks* a feature. By changing a toggle button or select menu near the feature name in the feature list a user can change the 3-way toggle to reserve, lock, or release the given feature. When the state of the feature is changed, it is then broadcast to all other users currently in the collaborative design session. This switch or button should visually represent the changed state of the feature (through color, font-style, etc.) and also provide information about which user in the collaborative design session has locked or reserved the feature. The mock-up example feature list in Figure 3-2 shows this toggle button as white when a feature is released, blue when a feature is reserved, and a lock icon when the feature is locked. The actual implementation of the state toggle should match the style of the CAx application into which it is integrated.

When a feature is reserved, it means that the user who reserved the feature is able to edit or delete the feature. A *reservation* can be made by any user at any time, even if the feature is already reserved by another user (although this does not hold true if the feature is locked by another user). Because a reservation can only be held by a single user at a time in a collaborative design session, it prevents two users from editing the same feature at the same time. A reservation could be thought of as a per-feature edit token, or a "soft" lock for edit permissions for a given feature. This allows for relatively open collaboration between users and allows them to work with a model collaboratively while preventing update and delete conflicts. The reservation system would work well for a collaborative design session involving relatively few people or involving a design that does not have strict design constraints/permissions about who can or cannot edit the model.

A locked feature is very similar to a reserved feature; the only difference is that when a feature is locked, the user who "owns" the lock must first release the lock before that feature can be reserved, constrained, or locked by any other users. Locking allows tighter control over features in a part, such as an interface feature with another model. The user interface could additionally implement convenience features to allow users to request a lock be released via chat message, email, icon transmit status, etc.

Depending on the current feature status (released/reserved/locked), feature items in the collaborative feature list form should be highlighted with a certain color. The mock-up feature list shown in Figure 3-2 displays released features as white, reserved features as blue, and locked features as gray, although any color scheme could conceivably be used. The highlighted color will visually convey the state of the feature to the user easily. A unique color for each user in the collaborative design session could also be used to convey which user has reserved or locked a given feature. For the feature list form, a one-click *state* toggle is recommended as it will make it simple for users to set the state of several features which need to be reserved or locked for editing. Additionally, the same color scheme could be applied to the rendered features inside the CAx system visual editor to help users see which features are edit-able (Figure 3-3). Decals/icons (such as a lock symbol) could also be applied to features in a CAx system's visual editor to further convey its current state to the user.

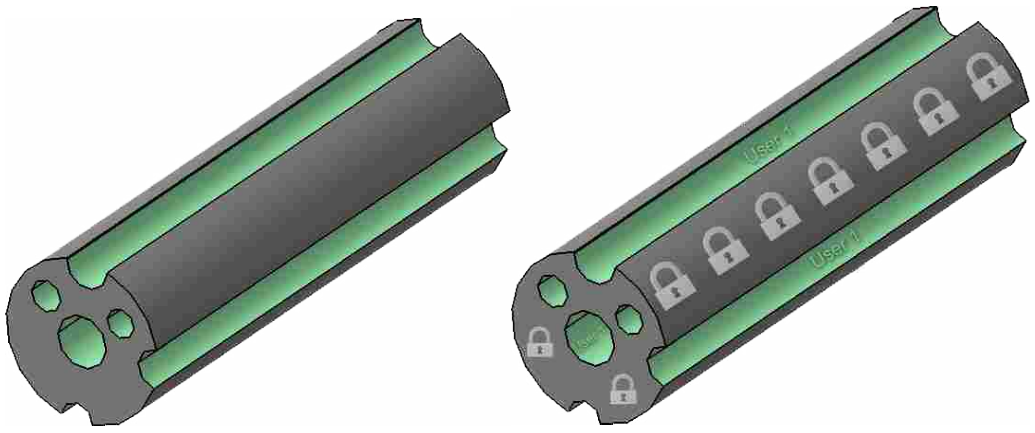


Figure 3-3: Colors and/or decals help users visualize the edit-ability of features

Additional user interface pieces could be used to toggle the state of features in the CAx visual editor as well. One method would be to use a right-click context menu as shown in Figure 3-4. Another method to toggle the state of a feature would be to simply use a triple-click of the mouse on that feature, which would allow users to quickly reserve and lock features while keeping their cursor in the visual editor. The ability to select a group of features either from the collaborative feature list or from the visual editor would also be very convenient to users who will be editing an entire group or section of features.

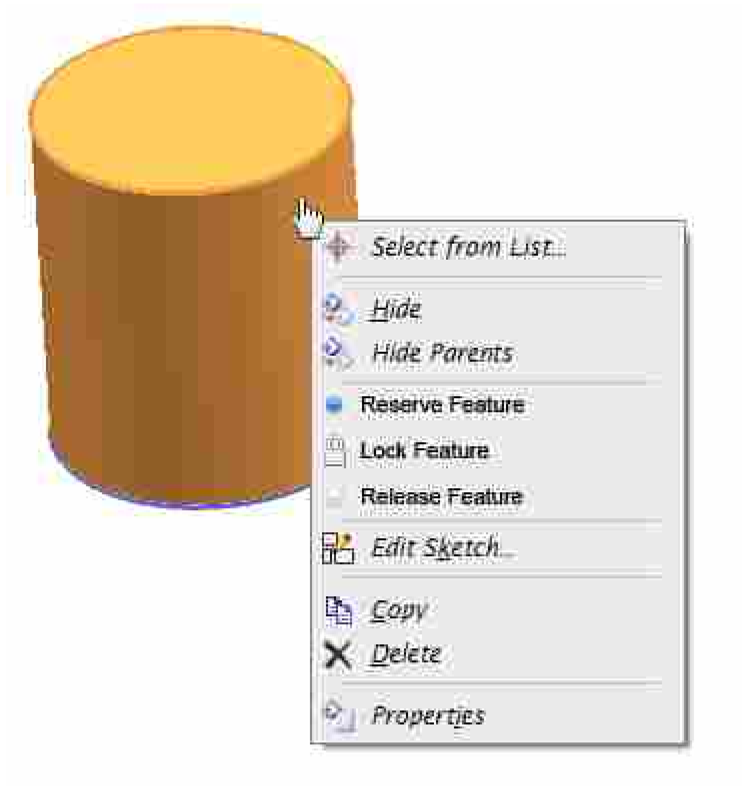


Figure 3-4: Mock-up of a right-click context menu being used to toggle the state of a feature

The ideal implementation of the collaborative feature list functionality would be to implement it directly into a CAx system's existing feature list using the same look and feel of that CAx system. Any other menus or buttons should be implemented similarly. This will make users familiar with that particular CAx software feel more comfortable with the collaborative features, and ultimately make it easier for them to learn and adopt the new features into their existing workflow.

This feature-wise locking approach also provides additional capability for CAx model decomposition. In addition to locking/reserving features individually, they could also be locked or reserved in groups. If features are related in a tree-like structure, an entire branch of features could be locked just by simply locking the parent-feature in the tree. As new features are added to a model, if they are connected to existing features in that tree or group, those features could in-turn inherit any lock/reservation properties from their parent feature. This functionality could provide a way to effectively partition a CAx model, having individual users working on individual groups of features in a model. It would also include less overhead than a spatial model decomposition method that decomposes a model using planes and surfaces. Figure 3-5 shows a mockup of how features could be locked in a tree-like structure. Feature 3 and all of its children are locked by User 1, and Feature 7 and all of its children are locked by User 2.

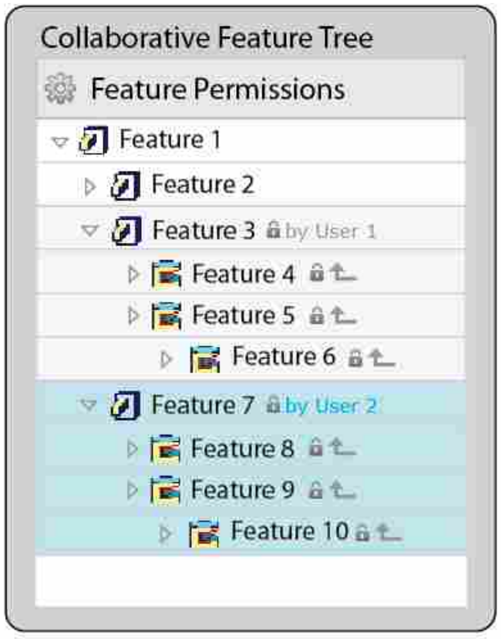
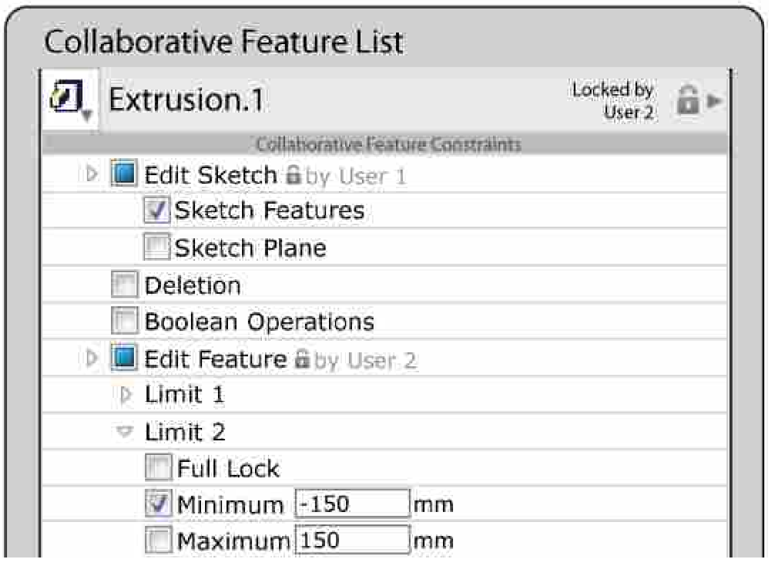


Figure 3-5: Mock-up of features locked by tree branch

3.3.2 Collaborative Feature Constraints

The advanced functionality of the collaborative feature list allows collaborative feature constraints to be added to the model. Collaborative feature constraints allow exact control over every aspect of the feature, which adds more flexibility to control the manipulation of the feature (compared to simply locking the feature). These constraints could be accessed from a collapsible or hide-able panel that can be accessed by users from the collaborative feature list using a single click. If a collaborative feature constraint is defined, the name of the user who created the constraint is displayed next to the constraint. Similar to locking, a convenience method could be implemented for a user to contact the "owner" of the constraint in the event that it needs to be modified or deleted.

Collaborative feature constraints will be specific to the type of feature that they govern and are based on the different data parameters that define that specific feature. For example, in a CAD system an extrusion is defined by a sketch (with its respective features [lines, arcs, etc.] and plane) as well as a starting limit, and an ending limit. Constraints for a CAD extrusion could constrain the "edit-ability" of the extrusion's sketch features or sketch plane, the ability to delete the feature, the ability to perform Boolean operations on the feature, as well as constrain the starting and ending limits to a certain value, or a certain range of values. Figure 3-6 shows collaborative constraints set for a CAD extrusion so that Extrusion.1's sketch feature cannot be edited, and that Limit 2 of the extrusion cannot be less than -150mm (also see Table 4-1). Collaborative feature constraint definitions will be unique for each different type of feature in a given CAx system and the best set of definitions will have to be determined depending on the implementation and the types of features in that CAx system.


Collaborative Feature List

Extrusion.1 — Locked by User 2

Collaborative Feature Constraints

- ▷ ■ Edit Sketch by User 1
  - [✓] Sketch Features
  - [ ] Sketch Plane
- [ ] Deletion
- [ ] Boolean Operations
- ▷ ■ Edit Feature by User 2
  - ▷ Limit 1
  - ▽ Limit 2
    - [ ] Full Lock
    - [✓] Minimum -150 mm
    - [ ] Maximum 150 mm

Figure 3-6: Mock-up of collaborative feature constraint schema for a CAD extrusion

3.3.3 User Awareness

Being aware of which other users are concurrently editing a document is an important feature of an effective collaborative design system. Hence, another important user interface element of this method includes a listing of the other users that are participating in the current collaborative design session (see Figure 3-7). Icons next to each user's name could indicate if they are active, inactive, busy, etc. similar to popular internet chat applications. Additionally, a chat bubble icon next to their name would allow users to initiate chat sessions between users as they are working together in the collaborative design session. This section would also be a place that other advanced communication features could be implemented such as audio/video chatting, or automatic translation of a text chat in a multi-lingual collaborative design session [19]. Clicking on a user's name could additionally reveal information about that user such as their expertise, location, position, company, etc. This piece of the collaborative user interface tools should be located near the collaborative feature list as there will likely be close interaction between these forms as users work with each other during the collaborative design session.

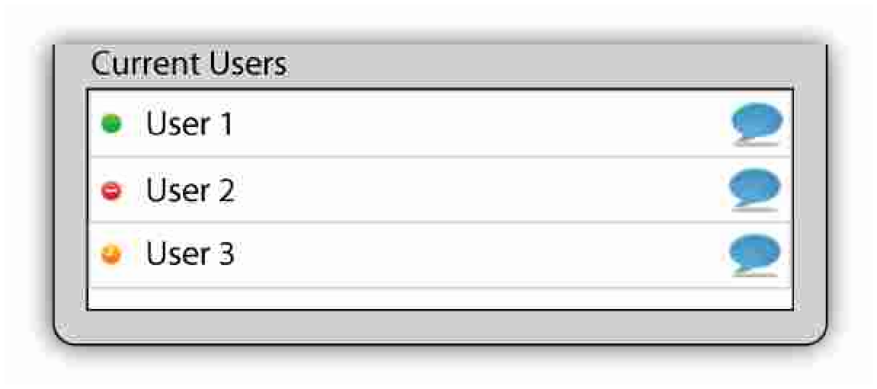


Figure 3-7: Mock-up of user awareness interface

3.3.4 Administrative Interface & Collaborative User Constraints

The administrative interface allows even deeper control over the flow of CAx data in a collaborative system, allowing administrative control over almost every aspect of the collaborative model. The administrative interface allows an administrator to control visibility and edit-ability of CAx data by user, by feature, by groupings of features (such as a part), by groupings of groupings of features (such as assemblies and subassemblies), and any combination of these groupings.

The administrative collaborative user constraint form as shown mocked-up in Figure 3-8 allows an administrator to enable or disable any number of different constraints for all the users in the system (see Figure 4-8 for an image of the implemented form). The functionality of these collaborative user constraints will vary depending on the implementation, but in this method recommended constraints include the ability to view a model, the ability to reserve features (and hence edit features), the ability to lock features, and the ability to create collaborative feature constraints. In the example below, in Figure 3-8 User 1 is allowed to view any CAD Model in the collaborative system (shown by the highlighted eye icon), as well as make reservations (shown by the highlighted blue dot icon). This user, however, is not allowed to create locks on any features (shown by the grayed-out lock icon), and is not allowed to create collaborative constraints (shown by the grayed out constraint icon). On this form, suggestions for usability would include tooltips that display when the cursor hovers over a certain option, which will display the option name, as well as the status for that option. It is also recommended that these options be toggled with a single click. Discussion of the enforcement of collaborative user constraints is found in section 3.5.

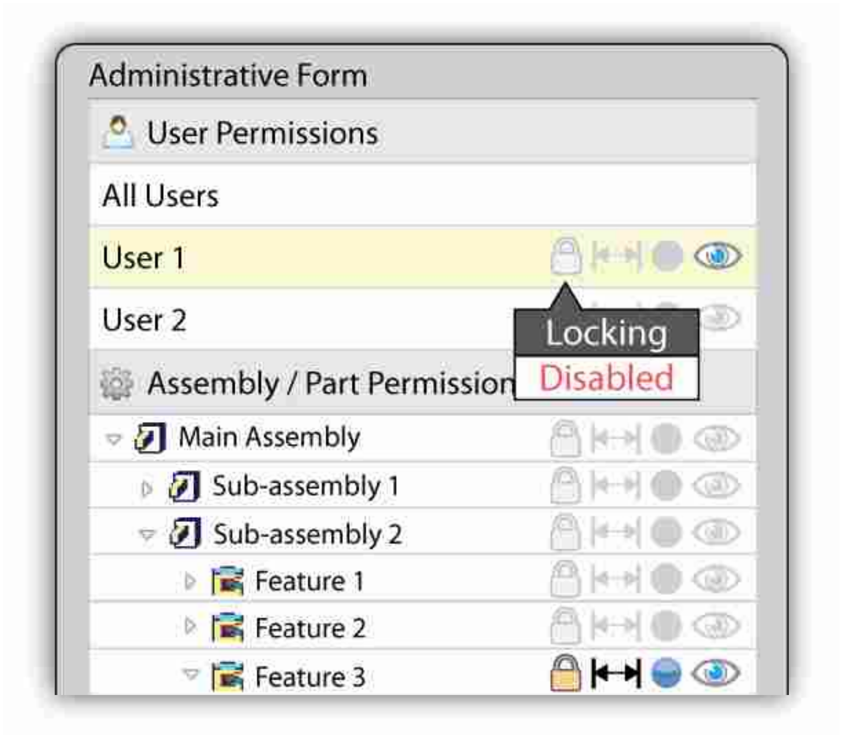


Figure 3-8: Mock-up of the administrative form

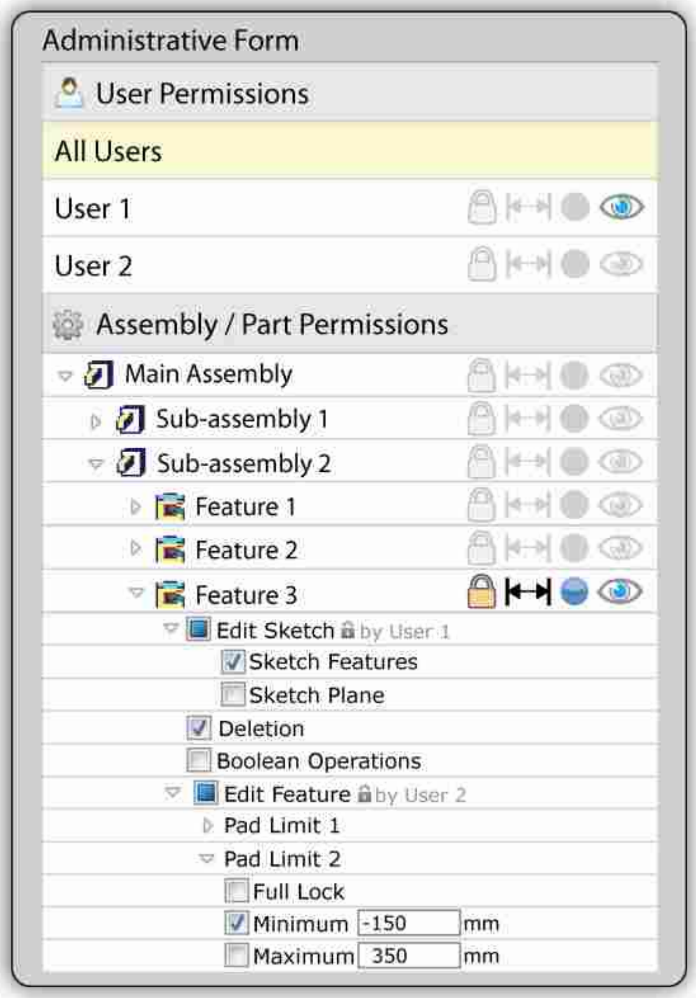
When a specific user's name is selected (highlighted in yellow) the permissions for this user are displayed and can be set per assembly, per sub-assembly, and per feature. When one of these options is toggled, the option could also be toggled for all children. For example, if locking for a certain user is enabled at an assembly level, then all of the features in all child sub-assemblies and parts could also have the locking enabled. The hierarchy (whether the global or user-specific constraints are enforced over one another) of these collaborative user constraints could also be modified to fit the needs of the specific application.

When the *All Users* option is selected, then the global collaborative constraints menu becomes active for all features (see Figure 3-9). In addition to creating global permissions for each assembly/sub-assembly/part/feature, this will allow the administrator to create or arbitrate any collaborative constraints that need to be modified. Global permissions for each assembly, sub-assembly, part, and feature will override the user-specific features. The hierarchy of the global constraints could also be modified to fit the needs of the specific application. Also, this form could potentially be used to manage task-based constraints or spatial model decomposition as suggested by Marshall [15] if such were also available in the same implementation.

As recommended with the previous user interface forms for this method, the administrative interface should be integrated directly into a CAx system using the same look and feel as is used for other administrative menus and settings windows. As the look and feel is kept consistent, it will make users feel more comfortable with the functionality that it provides.

3.4 Backend Organization and Operation

This section will detail the underlying data model and procedures that will be used to implement the above explained data consistency system.


Administrative Form

User Permissions

All Users

User 1

User 2

Assembly / Part Permissions

- Main Assembly
  - Sub-assembly 1
  - Sub-assembly 2
    - Feature 1
    - Feature 2
    - Feature 3
      - Edit Sketch by User 1
        - [✓] Sketch Features
        - [ ] Sketch Plane
      - [✓] Deletion
      - [ ] Boolean Operations
      - Edit Feature by User 2
        - Pad Limit 1
        - Pad Limit 2
          - [ ] Full Lock
          - [✓] Minimum -150 mm
          - [ ] Maximum 350 mm

Figure 3-9: Mock-up of administrative form with all users selected

3.4.1 Relational Database

It is recommended that for an implementation of this method that a relational database management system (or RDBMS), such as Microsoft SQL Server or MySQL, be used. A typical CAx modeling system's data is comprised of relational data which could easily be stored in an RDBMS system. To effectively manage all the collaborative feature constraints and their relations to the different features, parts, sub-assemblies, and assemblies, the RDBMS system would be ideal.

3.4.2 Collaborative CAx System Architecture

There are varied architectures that could be used to construct collaborative CAx design software implementations. For the method and implementation described in this thesis a client-server architecture was assumed, as this architecture is most adaptable to methods explained here. The methodology used here could also be implemented using other architectures such as peer-to-peer.

3.4.3 Generalized Data Model

A sample data-model representing a basic collaborative CAD system has been included in Figure 3-10. The underlying data-model of the data consistency method explained above could vary depending on the implementation, but this generalized data-model illustrates the concepts of how this method could be implemented.

The foundational piece of data of this data model is a group. All users, assemblies, and parts belong to a group. The *groups* table has an *admin_id* which is a foreign key specifying which user is the administrator for the group. The data-model could be slightly modified to allow several users in the group to have administrative privileges. Each assembly will have a *group_id* foreign key which relates it to its group. Sub-assembly nesting is created with the foreign key *assembly_id*. Each part can also be related to an assembly using its *assembly_id* foreign key. Each feature is related to a part through the *part_id*. This sample data model shows only one table for features, but in an actual implementation there would be several feature tables for each of the different kinds of features in the system. The data-model could be slightly modified to allow a many-to-many relationship between parts and assemblies as well, but for simplicity and purposes of this method it will be treated as a many-to-one relationship. The *user_constraints* table is related to users through the foreign key *user_id*. The *user_constraints* table is also designed to be related to assemblies, parts, and features through the columns *user_constraint_object_id*, and *user_constraint_type*. This table allows for user-specific constraints that can be made from the administrative form to be simply assigned to an object in any table that supports these permission options. The name of the object type that it is constraining goes in the *user_constraint_type* column, and the primary key of the object it is constraining goes into the *user_constraint_object_id* column. The format of the *user_constraints* table makes it very flexible and easily adaptable to new tables and new constraint types.

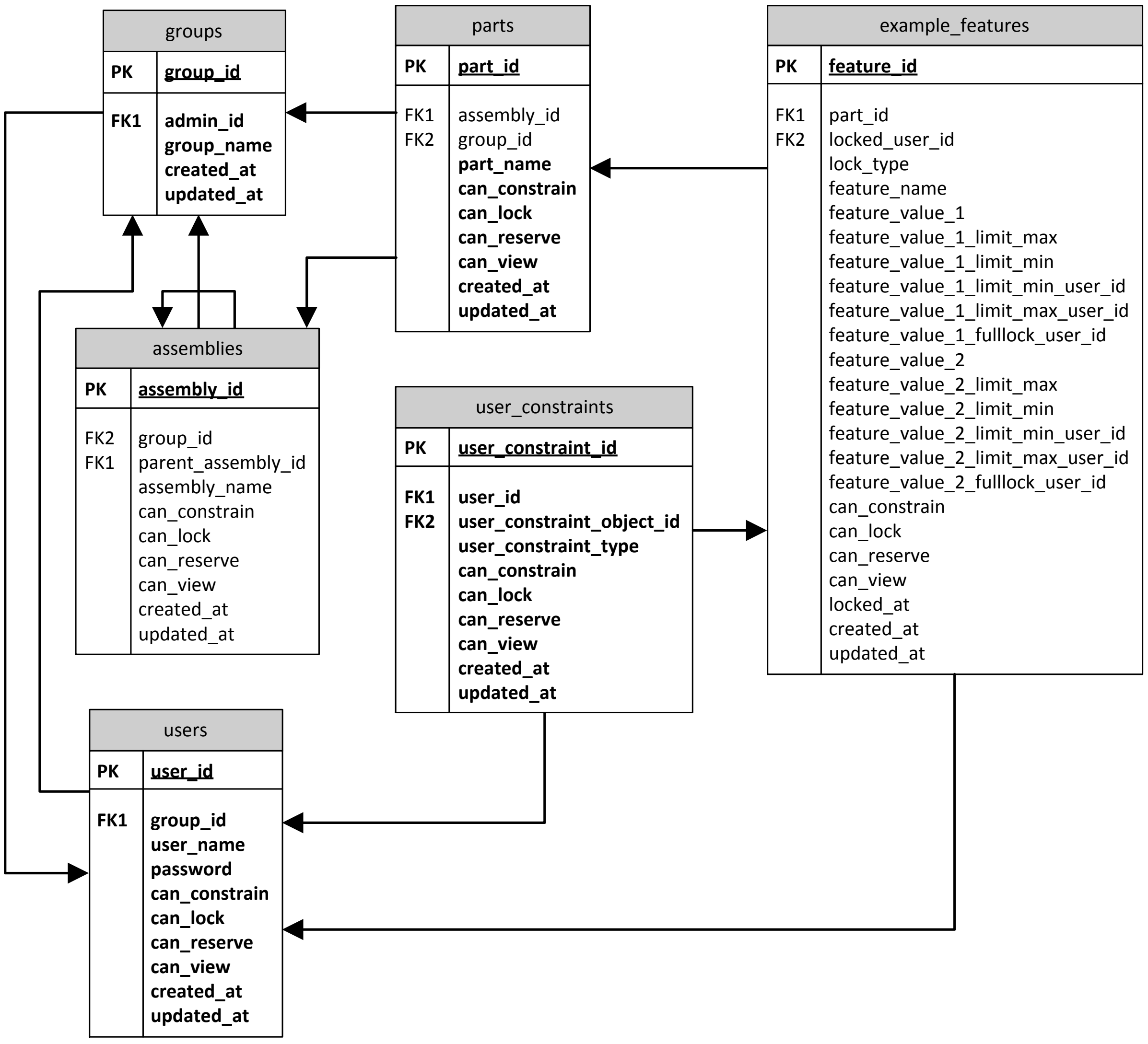


Figure 3-10: Sample data model

The data columns in the database tables *users*, *assemblies*, *parts*, and *features* are named *can_constrain*, *can_lock*, *can_reserve*, and *can_view*. These data columns are Booleans that enable or disable, respectively, the permissions to create collaborative user constraints, locks, reservations, or to simply view the object in question. Two different approaches could be implemented in the actual method implementation to determine which permissions override others, either having the parent permissions override the child permissions, or the converse. It is recommended that the child permissions override the parent permissions, as it would eliminate confusing logic that would have to be implemented in the user form to toggle child permissions as the parent permissions are changed. This approach is additionally more usable and intuitive for the user as they will not be as confused by which table's permissions ultimately govern the objects in question.

While this example data model only has one feature table, in an actual implementation of this method there would be several feature tables to represent the different kinds of features in the system. In a CAD model, there would be features such as extrusions, revolutions, sketches, planes, etc. which would all have different data relationships and data values needed to construct that actual feature. For example, an extrusion definition needs a related sketch, as well as a starting limit and ending limit. To enable locking and constraining of the different data values in the table, five additional columns are needed for each data value. These columns include maximum and minimum limits for each value, as well as a *user_id* that relates to the maximum and minimum limits to the user who created them. The column *feature_value_x_fulllock_user_id* relates a complete lock on that feature value to a user. If this column is set then that feature value will be completely locked. If the column is null then the feature value will be governed by any minimum/maximum limits set. If the minimum/maximum *user_ids* and values are not set for that feature value then no constraints will be active for that feature value.

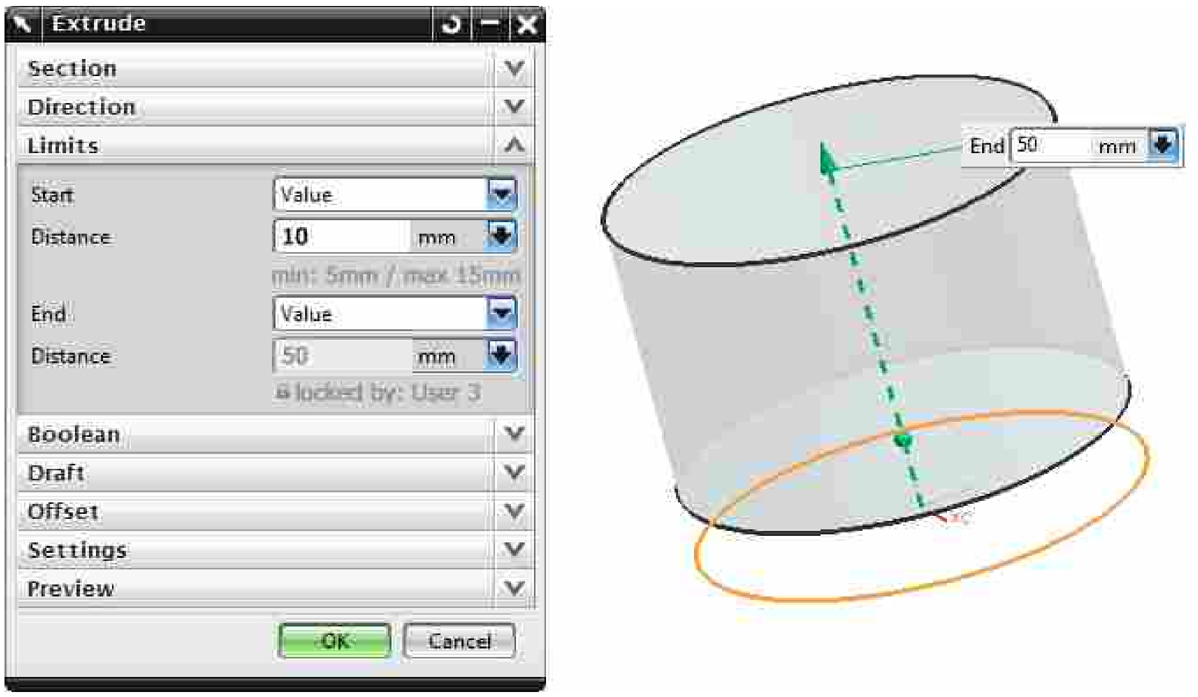
When a user reserves or locks a feature from the user interface, their *user_id* will be stored in the *locked_user_id* column for that feature. The *lock_type* column will then be set to either "reserved" or "locked" respectively. The current time the feature was reserved or locked will also be stored in the column *locked_at*. This information will be used to enforce locks and reservations. When a user releases the reservation or lock, the values *locked_user_id, lock_type*, and *locked_at* will be set to null.

It is to be noted that the example data model used for this section is extremely generalized for simplicity and understanding of this thesis. The data model and data relationships for an actual implementation of the described data consistency method would be more complete. The provided data model was only meant to be a starting point for an actual implementation.

3.5 Enforcing Reservations, Locks, and Constraints

The enforcement of reservations, locks, and collaborative constraints could be done in two different ways, depending on the implementation and API access for the given CAx system. For either method to be effective, it will be necessary to propagate modifications to any collaborative constraints or reservations/locks to each client as quickly as possible. With the information about these modifications each client program in the collaborative design session could effectively enforce the constraints.

The best and most intuitive way to enforce these data consistency constraints would be to implement the enforcement directly into the visual editor of the CAx software. If a feature was reserved or locked, or a collaborative constraint was set, then the visual editor would disallow the user from violating the feature locks or collaborative constraints. To be able to implement such capabilities would require either modification of the source code of the given CAx system, or API access to prevent editing of features. This method would be able to directly communicate to users from the native CAx visual editor the current state (released/reserved/locked) of each feature in the model, making the data consistency system more intuitive for users and more usable. A mock-up of what integrating collaborative feature constraints directly into a CAD software package (Siemens NX7) could look like is shown in Figure 3-11.



Start: Value
Distance: 10 mm
min: 5mm / max 15mm
End: Value
Distance: 50 mm
locked by: User 3
End 50 mm

Figure 3-11: Mock-up of collaborative feature constraint interface (Siemens NX 7)

If neither the software's source code nor sufficient API access is available to constrain users from editing the model from the standard CAx visual editor (as is the case in many commercial CAx software packages), then another approach could be used to enforce the locks and reservations: see Figure 3-12. This approach works by checking the edit permissions of each modified feature before modifications are propagated to the database. If the feature is not reserved or locked by the user trying to modify the feature, or any collaborative constraints are violated then the local modifications are not saved into the database, and the current CAx data is reverted to the pre-modified state. In some cases, it may be necessary to check the permissions of the parent feature as well when creating new features as some constraints, such as constraining the ability to perform Boolean operations on a CAD feature, may relate to that constraint. This enforcement approach is not as intuitive for users because the CAx visual editor will allow them to still edit the feature locally even if it is locked by another user. To help users understand how this works, a series of informative warnings would have to be displayed to notify the user how the data consistency system works and when feature locks or constraints have been violated. This enforcement method should also be implemented underneath of the first method described in this section as a failsafe to prevent modifications to locked/reserved/constrained features in the event that the collaborative constraint information is not immediately broadcast to all users. The process used to enforce collaborative constraints for feature modification is shown in Figure 3-12.

Enforcing collaborative user constraints that are not directly linked to feature modification can be done in a slightly different way. For example, to enforce a collaborative user constraint that prevents a certain user from viewing or opening a part or assembly, the process in Figure 3-13 could be used. Additionally, for the process of enforcing collaborative user constraints that prevent a user from reserving or locking a feature, the process in Figure 3-14 could be used. Finally, the process of allowing users to create collaborative users constraints can be found in Figure 3-15.

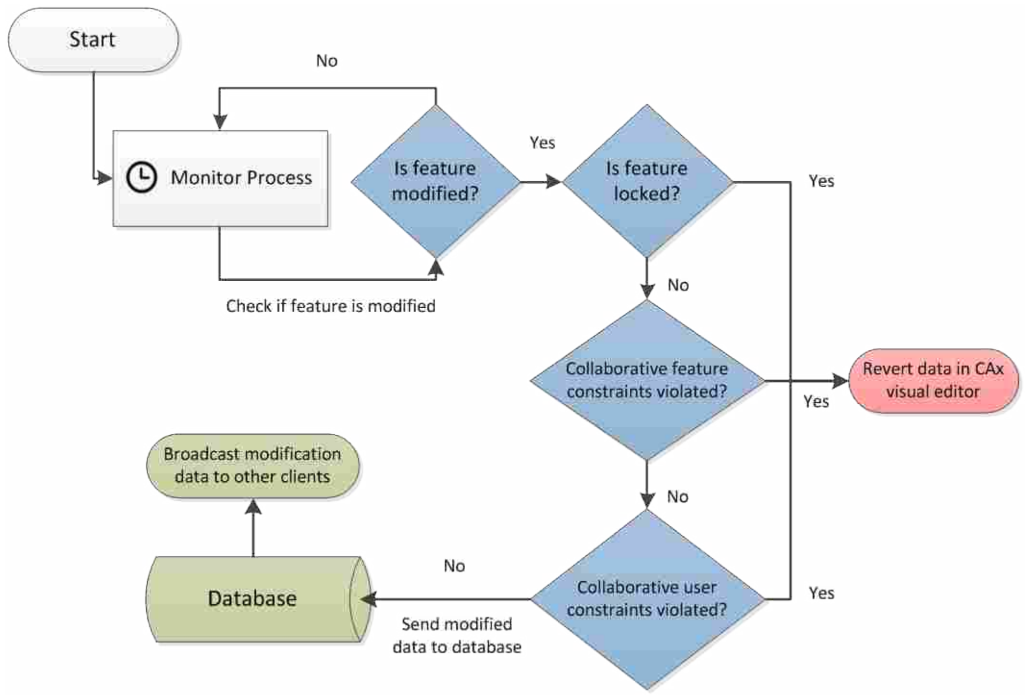

Start
No
Monitor Process
Is feature modified?
Yes
Is feature locked?
Yes
Check if feature is modified
No
Collaborative feature constraints violated?
Revert data in CAx visual editor
Yes
Broadcast modification data to other clients
No
Database
No
Collaborative user constraints violated?
Yes
Send modified data to database

Figure 3-12: Process showing the feature modification constraint enforcement

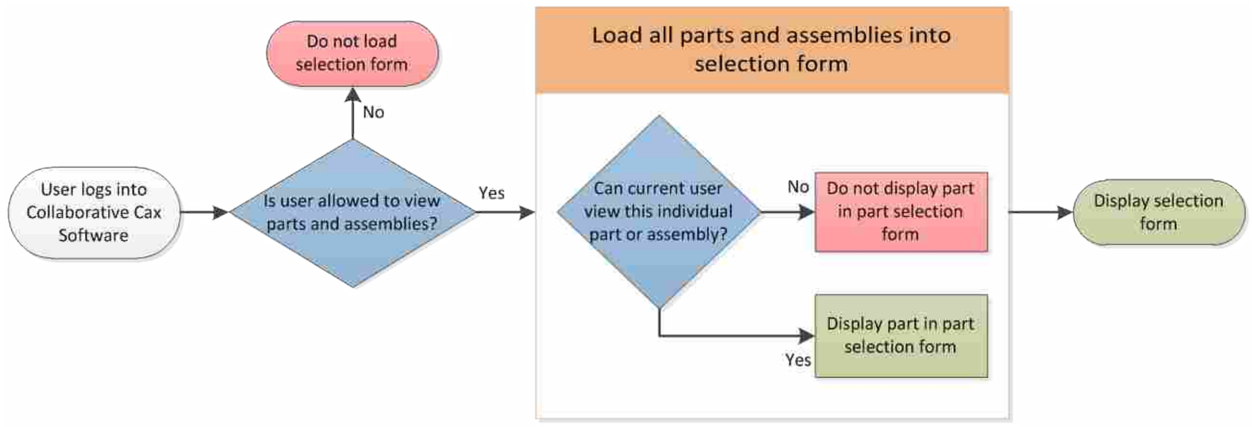

Do not load selection form
Load all parts and assemblies into selection form
No
User logs into Collaborative Cax Software
Is user allowed to view parts and assemblies?
Yes
Can current user view this individual part or assembly?
No
Do not display part in part selection form
Display selection form
Display part in part selection form
Yes

Figure 3-13: Enforcement of the a collaborative user constraint controlling model viewing permissions

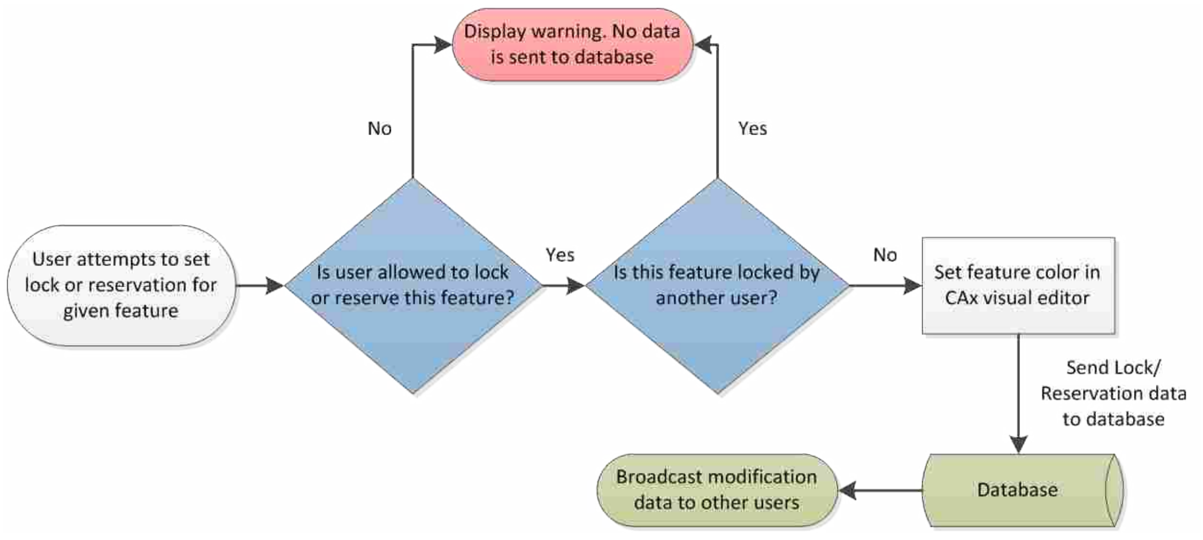

Display warning. No data is sent to database
No
Yes
User attempts to set lock or reservation for given feature
Is user allowed to lock or reserve this feature?
Yes
Is this feature locked by another user?
No
Set feature color in CAx visual editor
Send Lock/ Reservation data to database
Broadcast modification data to other users
Database

Figure 3-14: Process for enforcing collaborative user constraints for the ability to lock/reserve features

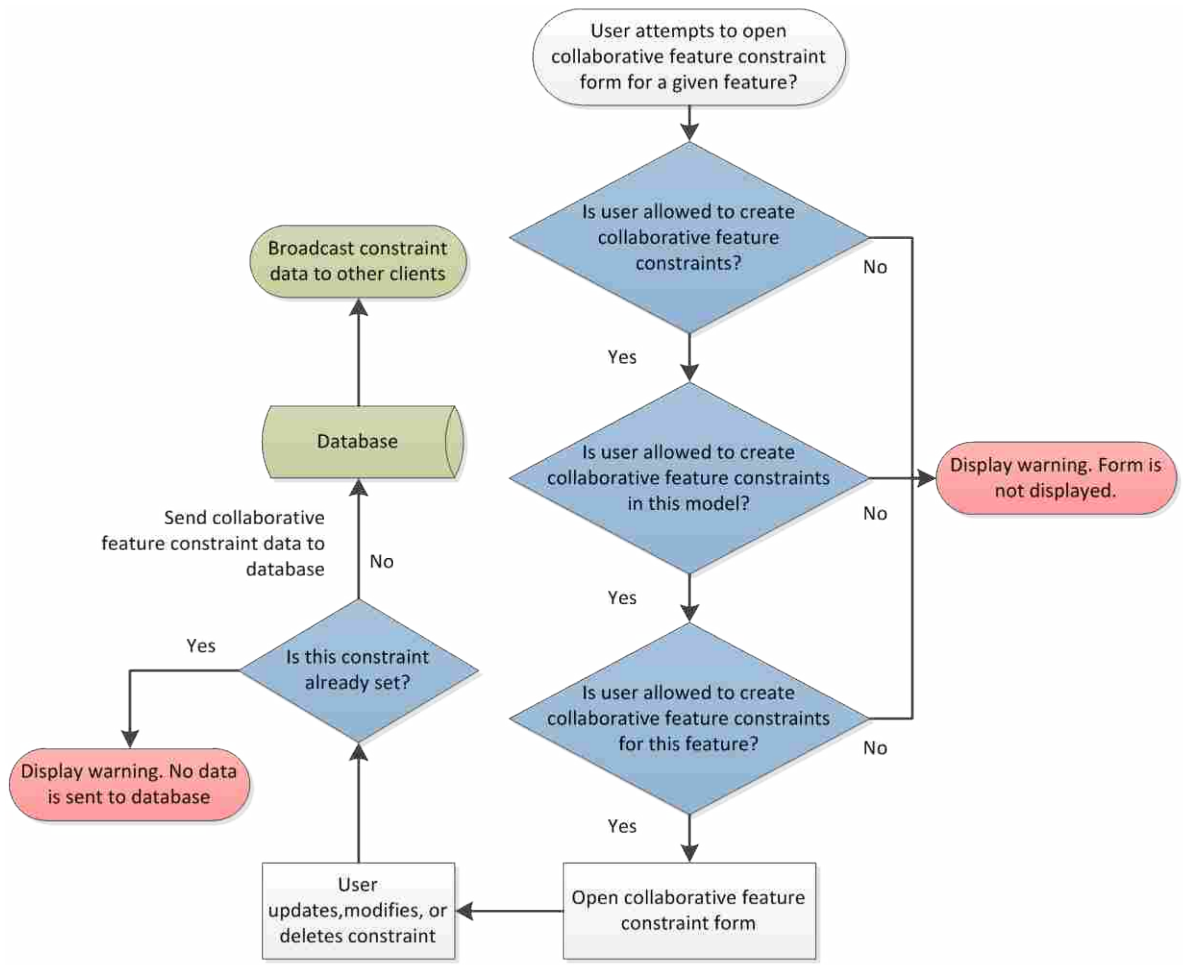

User attempts to open collaborative feature constraint form for a given feature?
Is user allowed to create collaborative feature constraints?
No
Yes
Is user allowed to create collaborative feature constraints in this model?
No
Yes
Is user allowed to create collaborative feature constraints for this feature?
No
Yes
Display warning. Form is not displayed.
Open collaborative feature constraint form
User updates,modifies, or deletes constraint
Is this constraint already set?
Yes
Display warning. No data is sent to database
No
Send collaborative feature constraint data to database
Database
Broadcast constraint data to other clients

Figure 3-15: Flowchart for enforcement of collaborative feature constraint creation

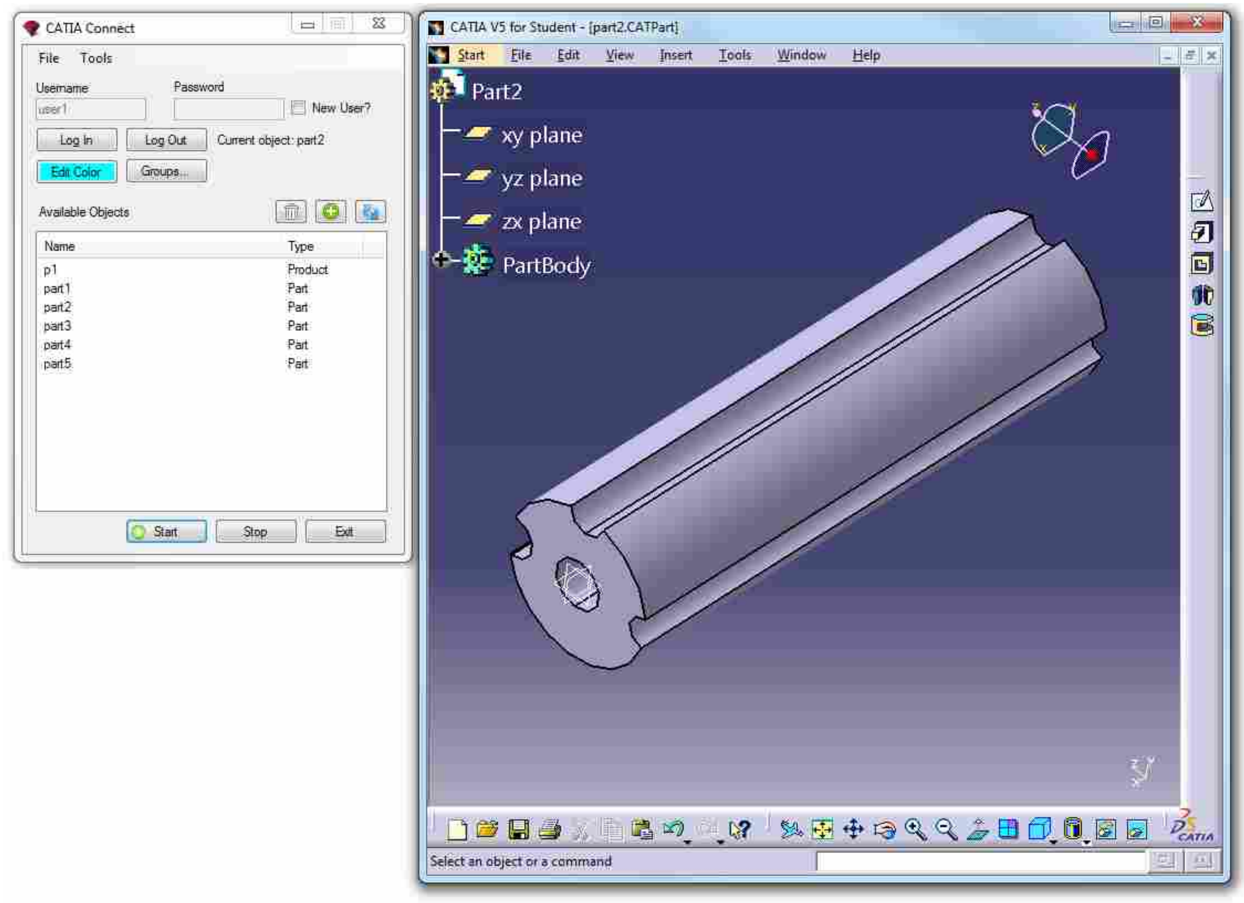
4 IMPLEMENTATION

4.1 CATIA Connect

The implementation of the method explained in Chapter 3 of this thesis was integrated into the CATIA Connect software plugin developed at Brigham Young University. CATIA Connect is a transparent adaptation multi-user CAD plugin that synchronizes CAD features between multiple clients. It was built using a client-server architecture which allows several users to participate in a multiple user design session simultaneously. The CATIA connect software is programmed to interface with CATIA CAD software using C# to interface with CATIA through its COM Object interface. COM objects are flexible objects that can be accessed by any .NET language such as Visual Basic, C++, or C# to manipulate data within an application. The COM API is able to extract CAD features from a CATIA session and store them in a Microsoft SQL Server 2008 database. The data defining each feature is then broadcast to all other clients in the same design session, and updated as features are created, modified, or deleted.

Because of limitations with the CATIA COM API such as a lack of event-driven call-backs and advanced control over the visual editor, the implementation is not completely seamless. New, modified, or deleted features for CATIA Connect are only pushed to the server every few seconds (rather than instantaneously) because there is no event interrupt in the CATIA COM API. A timer has been used in place of an event interrupt to push new geometry data to the server. Additionally new features are only retrieved from the server every few seconds as there is no server push system for the server to deliver new features to each client in the design session as they are created. A server push architecture such as that developed by Winn [20] could make retrieving new features from the server instantaneous and seamless, but is not implemented in this version of CATIA Connect. Additionally, CATIA connect does not implement user/part groups or assemblies as explained in section 3.4.3, and it can be assumed that all parts belong to one global group. The current implementation, however, is functional and has been created to a degree where it demonstrates a proof-of-concept of the functionality described in the method section of this thesis.

The CATIA Connect software was programmed using Visual Studio 2010 using C#. The user interface was built using the Visual Studio Windows Form Builder and is displayed as an external window to CATIA. Ideally the user interface for CATIA Connect and the data consistency method described in this thesis could be built directly into the CAx application using its native user-interface as recommended by Xu [19]. Direct integration would make the system more usable as it would feel more natural to users who were already familiar with the software. Due to a lack of API functionality to manipulate and integrate forms directly into CATIA, the user-interface has been implemented into forms external to the actual CAD system. Although these windows are not directly integrated into the CAD system they do demonstrate all the functionality needed to run the software effectively. The CATIA Connect form runs side-by-side with the CATIA visual editor (Figure 4-1). After a user is logged into CATIA Connect, a list of parts will be opened that can then be loaded. As the part is loaded, all the geometric features defining the part, as represented by data in the database will be created in the CATIA visual editor.



Username: user1
Password:
New User?
Current object: part2

Available Objects

Name	Type
p1	Product
part1	Part
part2	Part
part3	Part
part4	Part
part5	Part

Figure 4-1: CATIA Connect running along-side CATIA visual editor

As the part is loaded into the CATIA visual editor, the implementation will also launch the collaborative feature list form as well (Figure 4-2). From the collaborative feature list, the collaborative user constraints form is accessible for each feature listed in the feature list.

4.2 Data Consistency Method User Interface

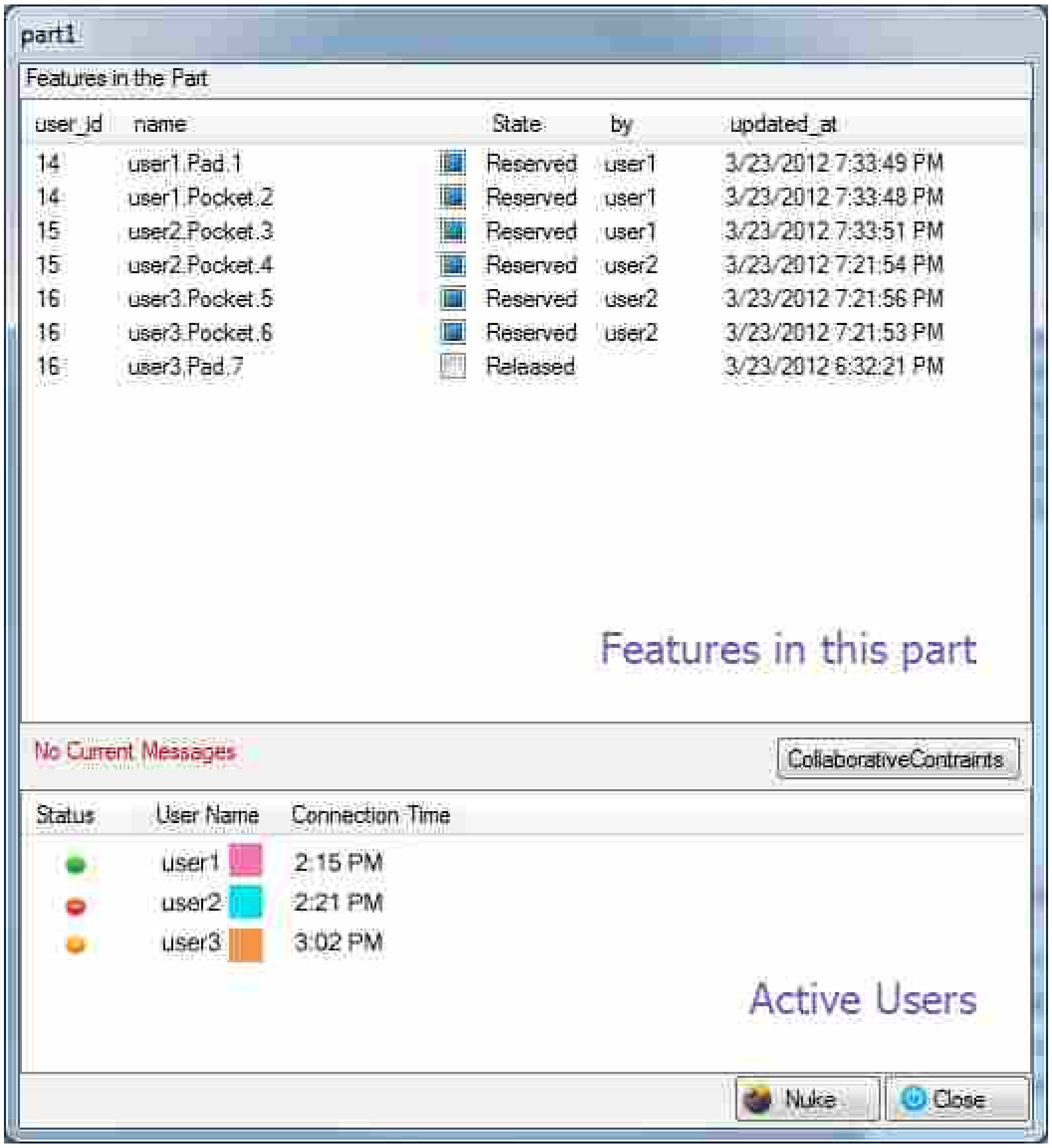
As the user interface was built, the main objective was to demonstrate the functionality of the data consistency system as a proof-of-concept. As such, this implementation has only been created to manage a small subset of the CAD features in CATIA Connect. Additionally, there are several improvements that could be made to enhance the usability of the system given additional time and resources. A commercial implementation (with more resources available) of the same method would be more robust, complete, integrated, and usable. The user interface for this implementation, however, will be functional and support as much functionality as possible given the available access of the CATIA API.

4.3 Feature List & Feature Reservation

4.3.1 User Interface

The collaborative feature list (see Figure 4-2) for this implementation has been built to mirror the feature list inside of the CATIA visual editor (a mock-up of this form is also available in Figure 3-2). As new features are added locally or on remote users' CATIA Connect sessions, they will be created in the CATIA visual editor and listed in the collaborative feature list. From the collaborative feature list, the lock/reserve/release functionality is very straight-forward to operate. There is a very simple three-state checkbox that can be toggled easily with one click. As the checkbox is clicked, the *state* column will update to display "released", "reserved", or "locked", and the *by* column will display the username of the user who has locked or reserved the feature (it will be left blank if the feature is released).

As the state of a feature is changed, the color of the feature is updated instantly in the CATIA visual editor to each user's color (see Figure 4-3) and changes to the database are propagated instantly. Remote clients will receive an update of the status change each time the update timer process in CATIA Connect retrieves updates from the database.


part1

Features in the Part

user_id	name		State	by	updated_at
14	user1.Pad.1	☑	Reserved	user1	3/23/2012 7:33:49 PM
14	user1.Pocket.2	☑	Reserved	user1	3/23/2012 7:33:48 PM
15	user2.Pocket.3	☑	Reserved	user1	3/23/2012 7:33:51 PM
15	user2.Pocket.4	☑	Reserved	user2	3/23/2012 7:21:54 PM
16	user3.Pocket.5	☑	Reserved	user2	3/23/2012 7:21:56 PM
16	user3.Pocket.6	☑	Reserved	user2	3/23/2012 7:21:53 PM
16	user3.Pad.7	☐	Released		3/23/2012 6:32:21 PM

Features in this part

No Current Messages

Status	User Name		Connection Time
●	user1	🟪	2:15 PM
●	user2	🟦	2:21 PM
●	user3	🟧	3:02 PM

Active Users

Figure 4-2: Feature list form

If a user tries to change the state of a feature that is locked by another user, a warning will be displayed, but the state of the checkbox will not be changed. If a user tries to change the state of a reserved feature, the state of the checkbox will simply change, the *state* label and the *by* label will be updated, and changes will be sent to the database immediately.

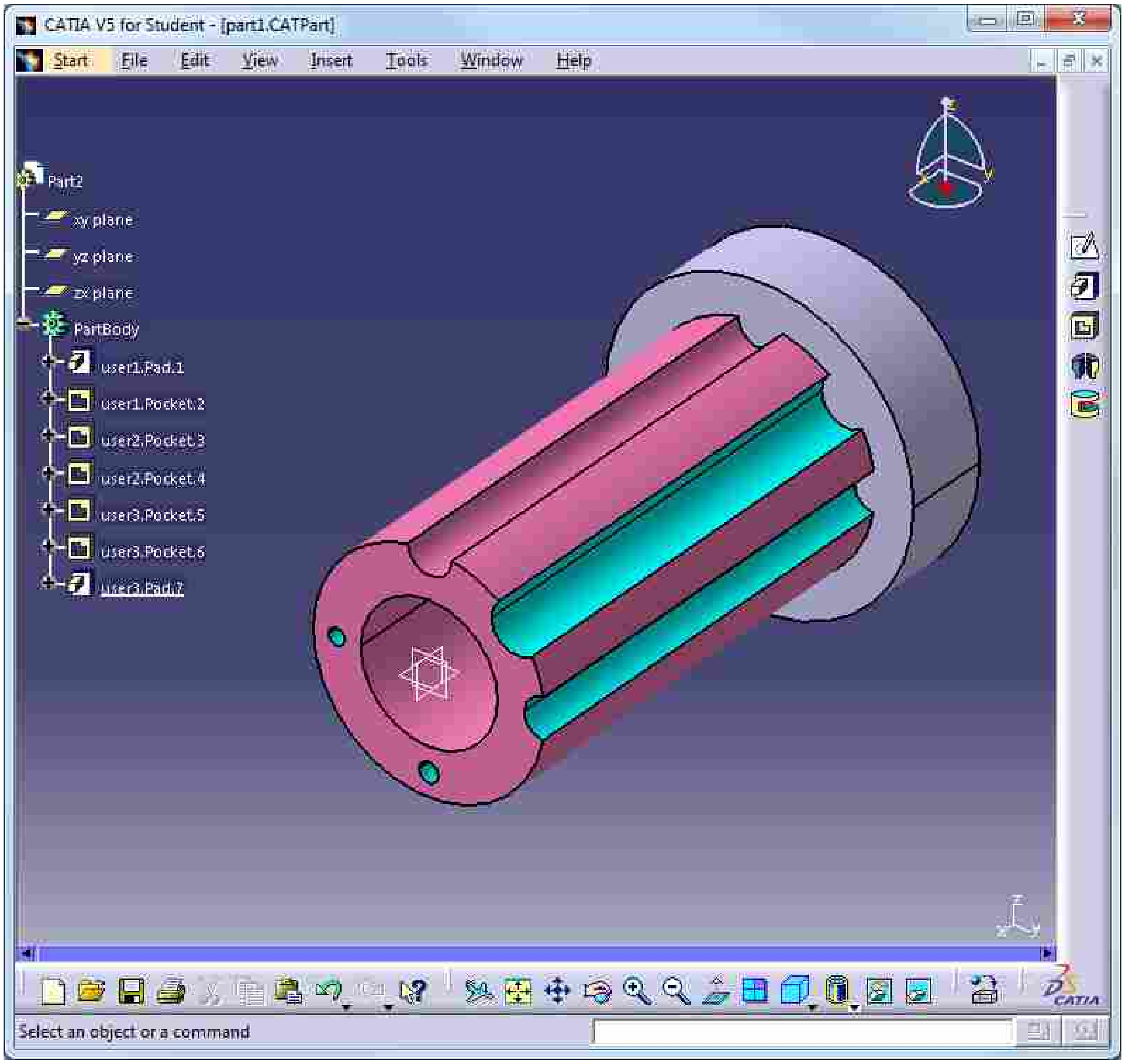


Figure 4-3: Colored CATIA CAD model representing the state of its locked/reserved/released features

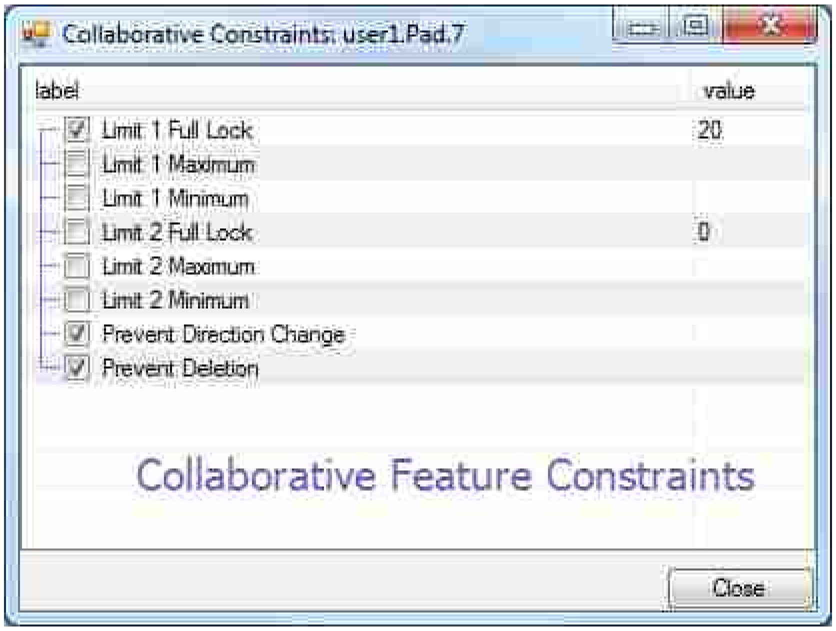
4.3.2 Enforcing Locked/Reserved Features

With the CATIA Connect API, there is no way to limit what a user can edit in the CATIA visual editor, so to enforce the unauthorized modification of features, a backend process must monitor the current features with each cycle of the CATIA Connect update timer. If this process finds a feature which is not locked or reserved by the current user, and has been edited, then the modifications from that feature will automatically be reverted to their previous state the next time that the update timer process checks the model. Additionally, a warning will be displayed to notify the user that they must reserve or lock the feature to edit it. If a feature is detected as modified and the user does have permission to edit this feature, then the changes will be sent to the database and broadcast to all other users in the collaborative design session. The enforcement of locked/reserved features is done in conjunction with the enforcement of collaborative constraints as shown in Figure 3-12 and explained in section 4.4.3 and 4.5.2.

4.4 Collaborative Feature Constraints

4.4.1 User Interface

The collaborative constraints menu can be opened from the feature-tree menu by double-clicking on one of the features or by selecting the feature and then clicking the *Collaborative Constraints* button (Figure 4-2). Once open, the collaborative constraints menu (Figure 4-4) is populated with the available collaborative constraints for the given features. The collaborative constraints can then be enabled/disabled by clicking the checkbox next to the label. The values for the collaborative feature constraints can also be edited directly from the form as well. The set of collaborative constraints available will differ from feature to feature as different features are defined using different parameter data. Full definitions for collaborative constraints for a CATIA pad (or extrusion) are found in Table 4-1.


Collaborative Constraints: user1.Pad.7

label	value
✓ Limit 1 Full Lock	20
Limit 1 Maximum	
Limit 1 Minimum	
Limit 2 Full Lock	0
Limit 2 Maximum	
Limit 2 Minimum	
✓ Prevent Direction Change	
✓ Prevent Deletion	

Collaborative Feature Constraints

Figure 4-4: Collaborative constraint form for a CATIA pad

Table 4-1: Collaborative constraint definition for a CATIA pad

Collaborative Feature Constraint Definitions for a CATIA Pad	
Limit 1 Full Lock	Completely locks the ability to change the limit 1 for the extrusion
Limit 1 Maximum	Limits the ability to make the limit 1 of the extrusion greater than the specified value
Limit 1 Minimum	Limits the ability to make the limit 1 of the extrusion less than the specified value
Limit 2 Full Lock	Completely locks the ability to change the limit 2 for the extrusion
Limit 2 Maximum	Limits the ability to make the limit 2 of the extrusion greater than the specified value
Limit 2 Minimum	Limits the ability to make the limit 2 of the extrusion less than the specified value
Direction Lock	Prevents the ability to change the direction of the pad
Deletion Lock	Prevents the ability to delete the pad

4.4.2 Collaborative Feature Constraint Management

Collaborative feature constraint definitions for each feature are managed by a set of classes in C#. A set of LINQ-to-SQL ORM (object-relational-mapper) objects are generated automatically from the database structure for CATIA Connect. There is a LINQ-to-SQL class created for each database table, which is also allowed to be extended through a partial C# class. These partial C# classes allow additional methods to be implemented for the feature object which allow a great way to manage and modify collaborative constraints. Additionally, another very simple class, collabconst, has been created to define collaborative user constraints, as shown in Figure 4-5. A list of all the collaborative feature constraints for a certain feature could be easily processed by a method in the corresponding partial LINQ-to-SQL class for that feature.

```
public class collabconst {
        public bool state;
        public string label;
        public double? value;
        public string valType;

        public collabconst(bool _state, string _label, double? _value, string
_valType) {
                state = _state;
                label = _label;
                value = _value;
                valType = _valType;
        }
}
```

Figure 4-5: Collaborative feature constraint object

The *collabconst* class contains four members and a constructor method. The *state* member defines if the collaborative feature constraint is enabled. The *label* member defines the text label that is displayed in the collaborative feature constraint menu for the constraint. The *value* member defines a certain value that is assigned to the constraint. This member would be used with a collaborative feature constraint containing a maximum or minimum limit but doesn't necessarily have to be used with all collaborative feature constraints. A constraint locking an entire feature value would not have to have a value defined, and if the value was defined then the value should not be editable. The *valType* member is used for convenience in converting values from the database to the proper format for the form in which they are being displayed.

To create a list of collaborative feature constraint definitions for a given feature, a method has been written in the partial c# class for each linq-to-sql object representing that feature. When called, this method will return a list of the current collaborative feature constrains active for the given feature object. Figure 4-6 shows the collaborative feature constraint definition set used in this way for this implementation for CATIA Pads. This method will typically be called when the collaborative feature constraint for is loaded for a feature and also when collaborative feature constraints are enforced for a feature. Additional work could be done to make the creation of collaborative feature constraints for additional features types more extensible, such as storing collaborative feature constraint definitions for each feature type in a database, but the approach explained above is effective for this implementation.

When a checkbox for any given collaborative feature constraint is checked in the form shown in Figure 4-4, first the permissions are checked to make sure that the user has permission to set a collaborative feature constraint, and that the collaborative feature constraint has not already been set by another user. If this check fails, a warning is thrown, and the constraint will not be set (similar to the process for reserving a feature shown in Figure 3-14). Otherwise, the corresponding *collabconst* object is then passed to the parent feature class, handled by a constraint processing method, and the data needed to set the constraint is immediately passed to the database. The collaborative feature constraint data is then broadcast to all other users in the collaborative design session and will be enforced by their local clients.

4.4.3 Collaborative Feature Constraints Enforcement

The enforcement of collaborative feature constraints is very straight-forward, and happens at the same time that the enforcement of locked/reserved features (as explained in section 4.3.2) takes place. After it is determined that a feature is editable (meaning the user has reserved or locked the feature), then each of the collaborative feature constraints are checked. If any of these constraints are violated, then the feature will be reverted to its original state in the CATIA visual editor and a warning message will be thrown: see Figure 3-12. Sample code enforcing collaborative feature constrains for a CATIA pad is provided in Figure 4-7.

4.5 Administrative Interface and Collaborative User Constraints

4.5.1 User Interface

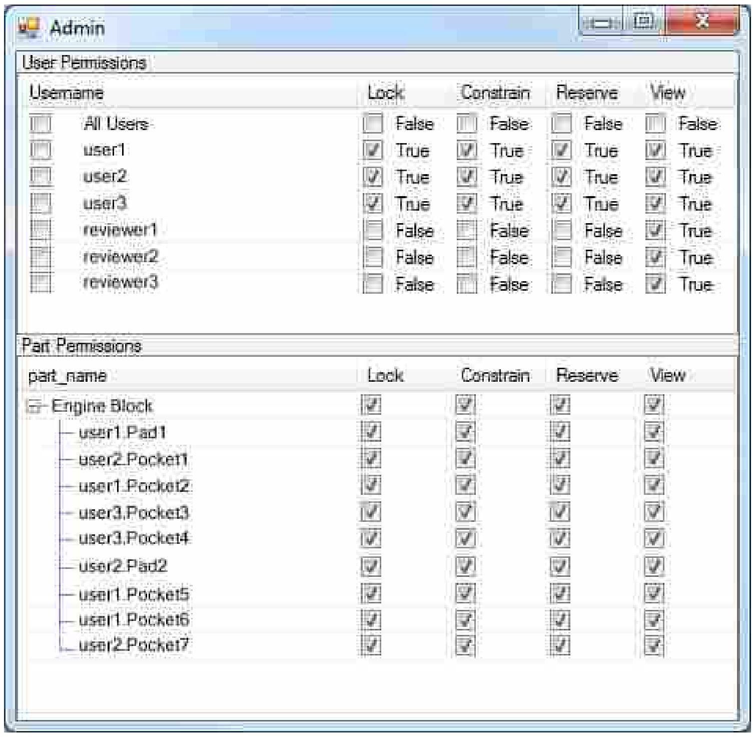
Collaborative user constraints are controlled by the administrative form (Figure 4-8) which can be opened only by CATIA Connect users who are designated as administrators (mock-up can be seen in Figure 3-9). As CATIA Connect does not implement user groups, administrators have global administration privileges. Having user groups that both parts and users belong to would be the ideal grouping unit for the administrative control, but this implementation does not employ such. The user permissions section of the administrative form controls the permissions set for each user as described in Section 3.3.4 of this thesis. The part permissions section of the administrative form allows the administrator to set the global permissions for each part, modify the collaborative feature constraints for each feature, or set administrative constraints for the currently selected user.

```
public List<collabconst> collaborativeConstraints() {
        List<collabconst> clist = new List<collabconst>();

        //Limit 1 full lock
        collabconst c1 = new collabconst(false, "Limit 1 Full Lock", null, "");
        c1.value = this.pad_limit_1;
        if (this.pad_limit_1_fulllock_user_id != null) c1.state = true;
        clist.Add(c1);

        //Limit 1 maximum
        collabconst c2 = new collabconst(false, "Limit 1 Maximum", null, "float");
        if (this.pad_limit_1_max_user_id != null) {
                c2.state = true;
                c2.value = this.pad_limit_1_max;
        }
        clist.Add(c2);

        //Limit 1 minimum
        collabconst c3 = new collabconst(false, "Limit 1 Minimum", null, "float");
        if (this.pad_limit_1_min_user_id != null) {
                c3.state = true;
                c3.value = this.pad_limit_1_min;
        }
        clist.Add(c3);

        //Limit 2 full lock
        collabconst c4 = new collabconst(false, "Limit 2 Full Lock", null, "");
        c4.value = this.pad_limit_2;
        if (this.pad_limit_2_fulllock_user_id != null) c4.state = true;
        clist.Add(c4);

        //Limit 2 maximum
        collabconst c5 = new collabconst(false, "Limit 2 Maximum", null, "float");
        if (this.pad_limit_2_max_user_id != null) {
                c5.state = true;
                c5.value = this.pad_limit_2_max;
        }
        clist.Add(c5);

        //Limit 2 minimum
        collabconst c6 = new collabconst(false, "Limit 2 Minimum", null, "float");
        if (this.pad_limit_2_min_user_id != null) {
                c6.state = true;
                c6.value = this.pad_limit_2_min;
        }
        clist.Add(c6);

        //Change Direction
        collabconst c7 = new collabconst(false, "Prevent Direction Change", null, "");
        if (this.can_modify_direction != null) c7.state = true;
        clist.Add(c7);

        //Prevent Deletion Direction
        collabconst c8 = new collabconst(false, "Prevent Deletion", null, "");
        if (this.can_delete != null) c8.state = true;
        clist.Add(c8);

        return clist;
}
```

Figure 4-6: Method for defining collaborative feature constraints for a CATIA pad in CATIA Connect

```
//Checking if the current user can update the feature
if(databasePad.isEditable(session.user_id)) {

  //Checking pad collaborative feature constraints limit 1 -----
  if(databasePad.pad_limit_1_fulllock_user_id != null) { //if limit locked
    //if the user changed a locked limit
    if(CATIAPad.FirstLimit.Dimension.Value != databasePad.pad_limit_1)
      databasePad.revertLimit1();//Reversing the changes to the CAD Mode

  } else {    //Checking the collaborative feature constraints

     // ----- Checking the limit 1 maximum -----
     if(databasePad. databasePad.pad_limit_1_max_user_id != null) {

         if(CATIAPad.FirstLimit.Dimension.Value > databasePad.pad_limit_1_max)
            CATIAPad.FirstLimit.Dimension.Value = (double)databasePad.pad_limit_1;

     // ----- Checking the limit 1 minimum -----
     if(databasePad.pad_limit_1_min_user_id != null) { //if constrained

        if(CATIAPad.FirstLimit.Dimension.Value < databasePad.pad_limit_1_min)
              CATIAPad.FirstLimit.Dimension.Value =
(double)databasePad.pad_limit_1;
        }

         // ----- Setting the database values to the new length -----
         databasePad.setLimit1(CATIAPad.FirstLimit.Dimension.Value);

     }


} else {      //Reverting changes if the part was not "editable"

       databasePad.revertAllData();
       session.myPartTree.setMessageLabel("You cannot edit " +
databasePad.pad_name + ". Please reserve or lock it to edit");


}

// ----- Updating the changes to the database -----
session.myPartTree.getConnection().SubmitChanges();
```

Figure 4-7: Code for collaborative feature constraints and feature locking enforcement for a CATIA pad



Admin

User Permissions

Username	Lock	Constrain	Reserve	View
☐ All Users	☐ False	☐ False	☐ False	☐ False
☐ user1	☑ True	☑ True	☑ True	☑ True
☐ user2	☑ True	☑ True	☑ True	☑ True
☐ user3	☑ True	☑ True	☑ True	☑ True
☐ reviewer1	☐ False	☐ False	☐ False	☑ True
☐ reviewer2	☐ False	☐ False	☐ False	☑ True
☐ reviewer3	☐ False	☐ False	☐ False	☑ True

Part Permissions

part_name	Lock	Constrain	Reserve	View
Engine Block	☑	☑	☑	☑
user1.Pad1	☑	☑	☑	☑
user2.Pocket1	☑	☑	☑	☑
user1.Pocket2	☑	☑	☑	☑
user3.Pocket3	☑	☑	☑	☑
user3.Pocket4	☑	☑	☑	☑
user2.Pad2	☑	☑	☑	☑
user1.Pocket5	☑	☑	☑	☑
user1.Pocket6	☑	☑	☑	☑
user2.Pocket7	☑	☑	☑	☑

Figure 4-8: Administrative form

4.5.2 Collaborative User Constraints Enforcement

When any one of the checkboxes for users, parts, or features (nested under parts) is clicked, the underlying user, part, or feature object (in this implementation it is just CATIA pads) will have a data member *can_lock*, *can_constrain*, *can_reserve*, or *can_view* set in the database. This info will be immediately broadcast to all users in the collaborative design session. These constraints will be enforced by each client locally. The *can_lock* and *can_reserve* constraints are enforced when a user tries to lock or reserve a feature from the feature list menu; if the user does not have the adequate permission, then a warning will be thrown and they will only be able to modify the lock status of the feature according to their permissions. The *can_constrain* constraint is enforced when a user tries to create or modify a collaborative user constraint; if the user does not have this collaborative user constraint enabled, then they will not be allowed to modify collaborative feature constraints, although they will still be able to view these constraints.

The *can_view* constraint will be enforced as the user tries to open a part; if the user does not have permission to open the part, then they will not be able to open it. Another method to enforce this constraint would be to simply hide the part from the user in the menu of available parts to open (as seen in Figure 4-1). Flowcharts for the enforcement of collaborative user constraints are found in section 3.5.

5 USAGE EXAMPLES, OBSERVATIONS, AND BENCHMARKING

5.1 Usage Examples

An example is provided in this section to illustrate how the developed method is effective and how it could be used with designers to enhance collaboration. The examples are all given in the context of a CAD system.

5.2 Engine Block Design Example

This example will demonstrate the functionality and the utility of the methods explained in this thesis including the following:

- Feature reservation/locking
- Collaborative feature constraints
  - Limit locking
  - Limit tolerances
  - Deletion prevention
- Collaborative user constraints
  - Part viewing
  - Feature reservation/locking
  - Permission to create collaborative feature constraints

This example starts with the objective of three designers (User1, User2, and User3) modeling the engine block seen in Figure 5-1 using CATIA Connect.

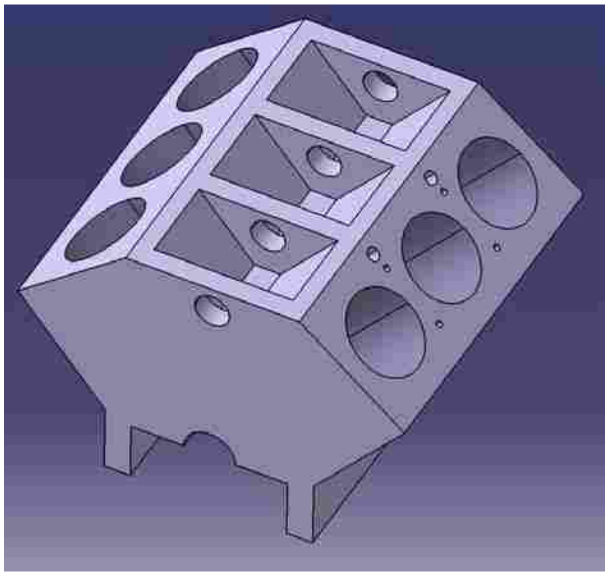


Figure 5-1: Engine block to design

For the first modeling operation, User1 creates the main engine block (Figure 5-2). This user also places a *deletion lock* collaborative feature constraint, to prevent accidental deletion of this key feature by other users.

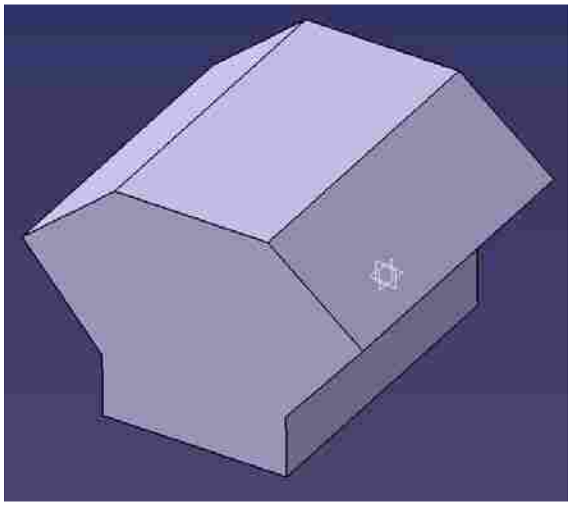


Figure 5-2: First step of the engine block modeling

Immediately after the block is created, User1 and User2 are able to simultaneously create several negative extrusions to continue the shaping of the engine block (Figure 5-3).

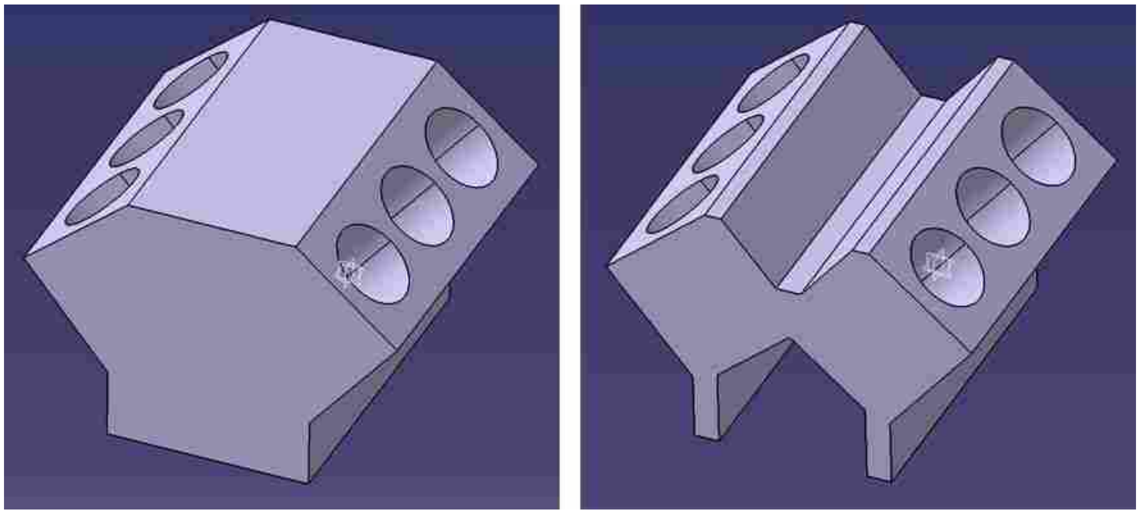


Figure 5-3: Next two steps

User1 then locks the negative extrusions that will be used for the piston cylinders. These extrusions then immediately turn User1's color, which is blue, communicating to the other users that this particular feature is locked by User1, and that it cannot and should not be edited (Figure 5-4).

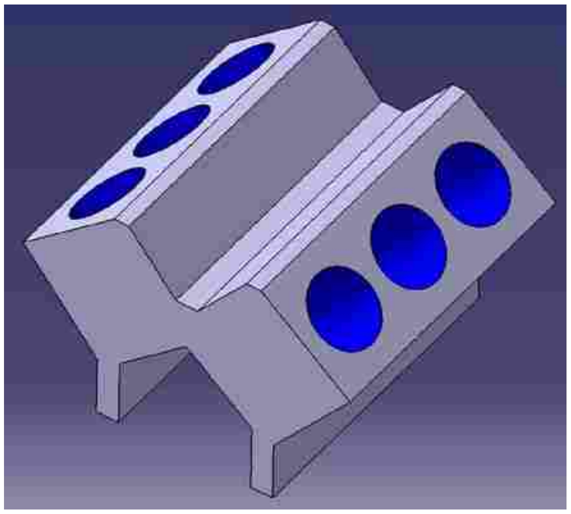


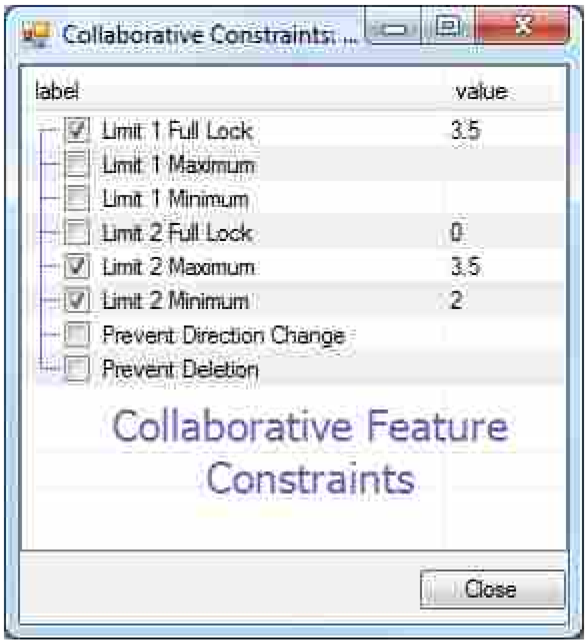
Figure 5-4: Piston cylinders are locked. (The blue color reflects their current state).

As soon as the cylinders for the pistons are created, User3 is able to begin the modeling of additional negative extrusions around the piston cylinders to reduce weight and dissipate heat

(Figure 5-5). After creating this feature, User3 needs to place three collaborative feature constraints on it. In order to do so, User3 must ask User1 (who is the administrator) to enable the collaborative user constraint that allows this. User1 enables this collaborative user constraint using the Administrative Form. After having this collaborative user constraint enabled, User3 is able to create the following collaborative feature constraints:

- Limit 1 full lock
- Limit 2 maximum of 3.5 inches.
- Limit 2 minimum of 2 inches.

These collaborative feature constraints are created to keep the first limit of the extrusion flush with the angled surface of the engine block, and to keep the depth of the extrusion within a certain allowable tolerance. These collaborative feature constraints will allow this feature to be adjusted according to heat-transfer and stress optimization analysis results, while preserving other geometry that should not be modified.


Collaborative Constraints: ...
Collaborative Feature Constraints
Close

label	value
[✓] Limit 1 Full Lock	3.5
[ ] Limit 1 Maximum	
[ ] Limit 1 Minimum	
[ ] Limit 2 Full Lock	0
[✓] Limit 2 Maximum	3.5
[✓] Limit 2 Minimum	2
[ ] Prevent Direction Change	
[ ] Prevent Deletion	

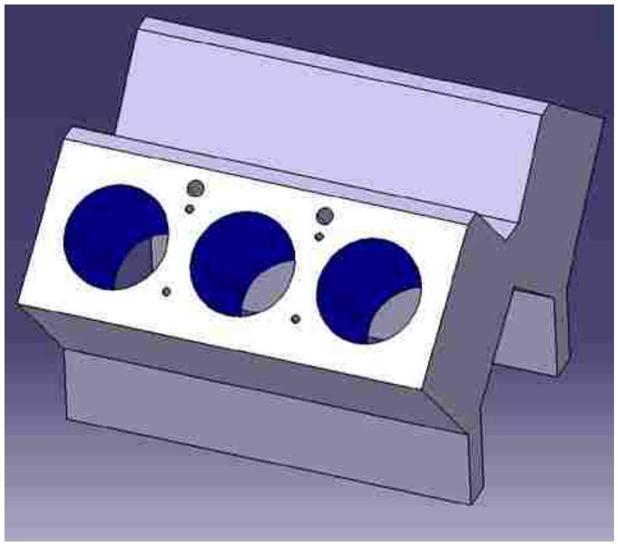
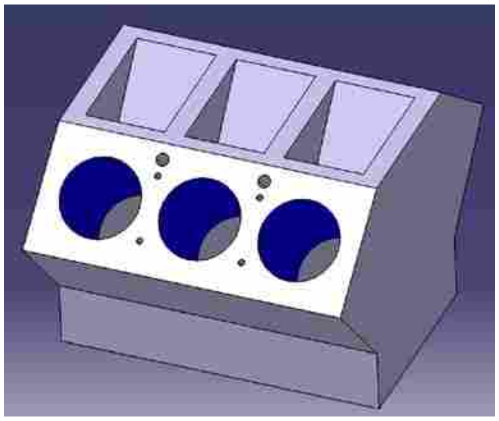
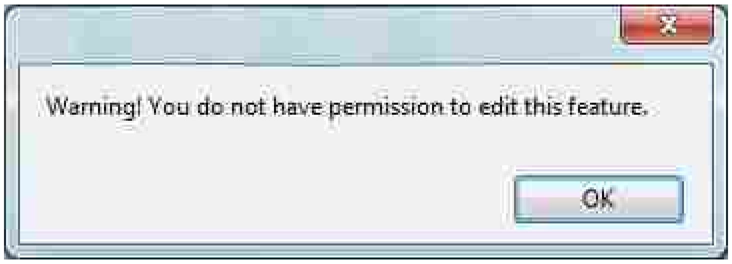


Figure 5-5: Creating collaborative feature constraints for negative extrusions for heat dissipation

The modeling of the engine block continues as User2 simultaneously creates 4 ribs which are formed through the entire block (Figure 5-6). While User2 is creating the ribs, User3 makes a small mistake, and attempts to edit the piston cylinders. Luckily this feature was previously locked by User1 so the change that User3 made was not allowed by CATIA Connect, preserving the geometry of that feature. A warning was also displayed to notify User3 that they do not have permission to edit this feature.





Warning! You do not have permission to edit this feature.

Figure 5-6: Creation of rib features and edit permissions warning dialog box

Once the ribs are created, User1 creates additional negative extrusions in the engine block for the camshaft and the crankshaft (Figure 5-7).

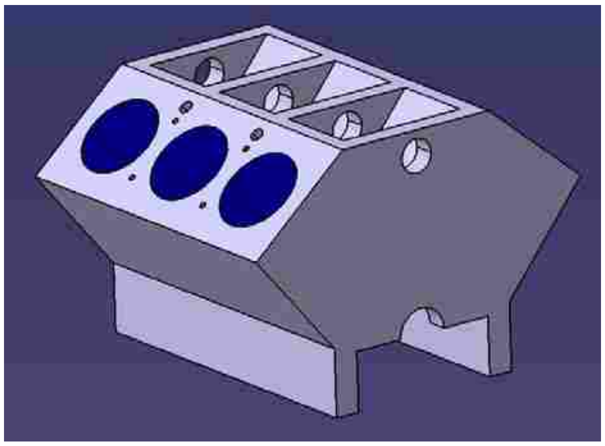


Figure 5-7: Negative extrusions for camshaft and crankshaft

Next, analysis results are received from an analysis group via email that is sent to User1, User2, and User3 at the same time. These results specify that the optimal value for the outer piston hole extrusion limit 2 is 2.8 inches. As these results are received, all three users are eager to update the model according to these results, and all attempt to reserve the feature to make the appropriate modifications at the exact same time. As each user tries to make the reservation, User1 is the user who is able to secure the reservation, and ultimately is the user who is able to make the modification to this feature. As User2 and User3 see that the feature is reserved by User1, they immediately know that User1 will be making the appropriate changes to that feature. This also spurs a short chat conversation where User1 explains to the other two users that the appropriate changes to this feature have been made, and clears up any confusion about who will make the design changes.

After this situation is resolved, the final features are created to complete the engine block design. The larger inner holes for the crankshaft are created by User2 (Figure 5-8 left), while User3 is able to concurrently create the final rectangular slot to finish the geometry of the engine block (Figure 5-8 right).

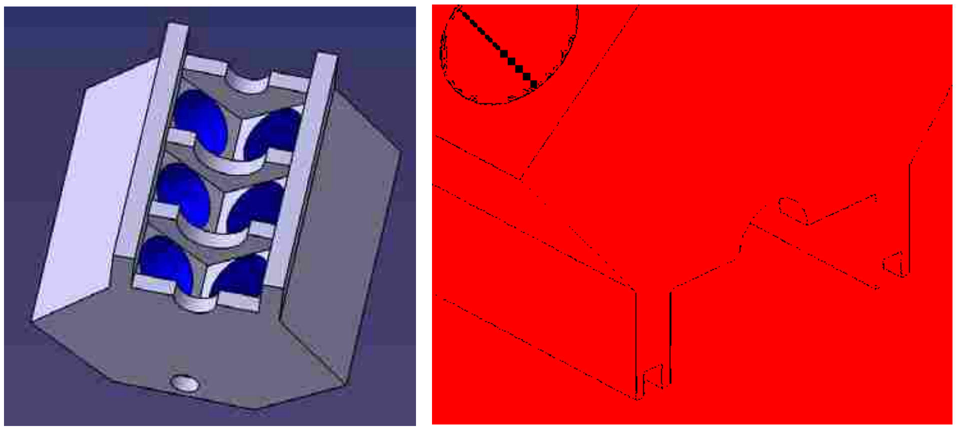
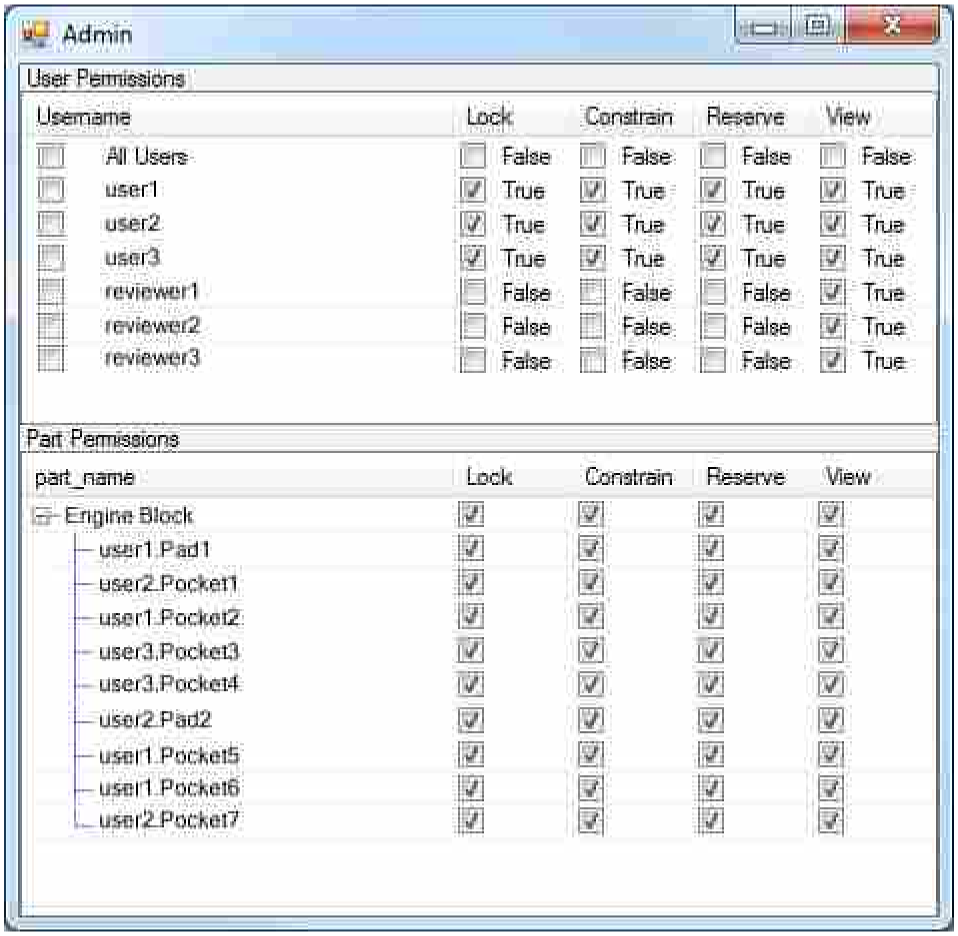


Figure 5-8: Additional crankshaft geometry

Now that the geometry for the engine block is complete, a design review will take place. For this design review, the geometry for this part will be shared with several reviewers, some which are remotely connected. To allow the reviewers access to the part, User 1, who is also an administrator, enables the *can_view* collaborative user constraint for them. Additionally, to protect the created feature data for this part, the reviewers will be restricted to a view-only mode using collaborative user constraints (Figure 5-9). The collaborative user constraints for the reviewers are enabled to prevent accidental or unauthorized modification of the model.



User Permissions

Username	Lock	Constrain	Reserve	View
All Users	False	False	False	False
user1	True	True	True	True
user2	True	True	True	True
user3	True	True	True	True
reviewer1	False	False	False	True
reviewer2	False	False	False	True
reviewer3	False	False	False	True

Part Permissions

part_name	Lock	Constrain	Reserve	View
Engine Block	✓	✓	✓	✓
user1.Pad1	✓	✓	✓	✓
user2.Pocket1	✓	✓	✓	✓
user1.Pocket2	✓	✓	✓	✓
user3.Pocket3	✓	✓	✓	✓
user3.Pocket4	✓	✓	✓	✓
user2.Pad2	✓	✓	✓	✓
user1.Pocket5	✓	✓	✓	✓
user1.Pocket6	✓	✓	✓	✓
user2.Pocket7	✓	✓	✓	✓

Figure 5-9: Admin form showing collaborative user constraints for the engine block

As the design review takes place, two design changes are recommended by the reviewers. It is recommended that the slot on the bottom of the block is extended another two inches, and it is recommended that another similar slot be placed in the other side of the engine block. The first change is made by User2 by reserving the slot feature—turning it a green color—then extending it two inches (Figure 5-10). The second change is made simultaneously by User1 who creates the new slot feature (Figure 5-10). These changes are able to be made quickly by User1 and User2 and the reviewers are able to approve of the design changes in real-time. The design review is now complete, and now the design of the part is done (Figure 5-1).

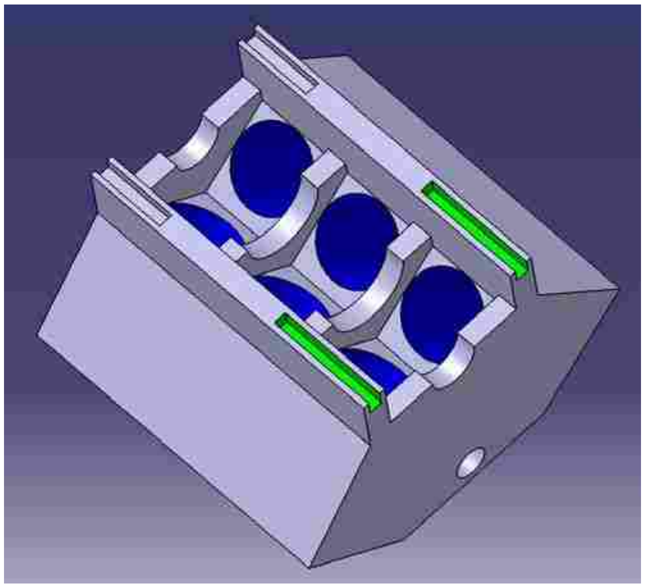


Figure 5-10: Changes implemented from design review feedback

5.3 Functional Observations

Additional validation for this implementation was done with a test run with several users in a multi-user design session with CATIA Connect. In this test, several users in the multi-user design session were challenged to edit several different features (Figure 5-11) that were either reserved or locked. After several minutes, the test was stopped, and the original feature data from the CAD model was compared to the final feature data of the CAD model. In all cases that this test was performed, the original data was exactly the same as the final data of the CAD model (Table 5-1, Table 5-2). This test demonstrates the functionality of this implementation effectively in preventing unauthorized changing of feature data.

Figure 5-11: Test part with several features

Table 5-1: Starting model data

pad_id	name	direction	limit_1	limit_2	created_at	updated_at
128	rob.Pad.1	0	25.4	0	21:13:52.5	21:57:12.420
129	rob.Pad.2	0	76.2	0	21:13:52.5	21:57:11.320
130	rob.Pocket.3	1	76.2	0	21:13:52.5	21:57:11.919
131	rob.Pad.4	0	127	0	21:13:52.5	21:57:10.680
132	rob.Pocket.5	1	127	0	21:13:52.5	21:57:09.202
133	rob.Pad.6	0	50.8	0	21:13:52.6	21:57:10.121
134	rob.Pocket.7	1	50.8	0	21:13:52.6	21:57:08.567
135	rob.Pad.8	0	50.8	0	21:13:52.6	21:57:07.923

Table 5-2: Final model data after several users attempted to edit data over a several minute design session

pad_id	name	direction	limit_1	limit_2	created_at	updated_at
128	rob.Pad.1	0	25.4	0	21:13:52.5	22:03:35.473
129	rob.Pad.2	0	76.2	0	21:13:52.5	22:03:36.143
130	rob.Pocket.3	1	76.2	0	21:13:52.5	22:03:36.613
131	rob.Pad.4	0	127	0	21:13:52.5	22:03:37.440
132	rob.Pocket.5	1	127	0	21:13:52.5	22:03:37.880
133	rob.Pad.6	0	50.8	0	21:13:52.6	22:03:38.363
134	rob.Pocket.7	1	50.8	0	21:13:52.6	22:03:39.270
135	rob.Pad.8	0	50.8	0	21:13:52.6	22:03:39.690

5.4 Benchmarking

As research for this thesis has developed and other CAx model decomposition methods have been reviewed, it has become evident that one decomposition method will not be perfect for every CAx design situation. Depending on the type, size, shape, interfaces, or complexity of the model, in addition to the number of users and the way these users interact with the model, different decomposition methods may be most effective for different situations. As different data consistency systems and model decompositions are implemented into multi-user CAx system, each design project will have to decide which method is most effective, or in some cases if a hybrid method is most effective.

One method that could be effective in comparing different data consistency and model decompositions includes using a simple formula. This formula will help calculate a relative productivity rate, P, for each method which could give users an idea of which method would be best to use. It is recommended that a single-user CAx system be used as a baseline comparison, in which the productivity rate would equal 1, i.e., P = 1.

The ideal productivity rate for a multi-user CAx system with n users could then be defined as P = n. The production or design time would roughly reduce proportional to the inverse of P.

In practice there will certainly be inefficiencies and overhead in managing a multi-user model and some productivity will be lost as users coordinate their design efforts. To determine the actual productivity of the multi-user method, adjustments can be added to the equation to account for inefficiencies. The first type of adjustment is a user-dependent "friction-factor-like" adjustment factor, μ, which accounts for system inefficiencies that grow as additional users participate in the multi-user design session. Examples of such inefficiencies could include the additional time and effort to lock or unlock features as edits are made, time spent in increased communication while design changes are made, or time spent waiting for the current model to synchronize. The second type of adjustment is a productivity overhead factor, o, which is independent of the number of users participating in the collaborative design session. Examples of productivity overhead could include time spent setting up collaborative user constraints, creating model decomposition, etc. The productivity rate could then be determined as follows:

$$P = n - (\mu_1 + \mu_2 + \cdots + \mu_i)n - (o_1 + o_2 + \cdots + o_j) \qquad 5.1$$

where i is the number of user-dependent efficiency adjustments, and j is the number of user-independent efficiency adjustments. Calculated productivity rates for different CAx design projects using different decomposition and data consistency methods could then be compared.

This thesis will not try to calculate or define the efficiency adjustment factors as there could be so much variability in their values, and likely would have to be determined experimentally on a project-by-project basis. This benchmarking method is rather presented to help designers estimate and determine which data consistency and model decomposition method will be best for their multi-user CAx project. Regardless, using such a benchmarking technique could feasibly help users and companies select the best multi-user data consistency and decomposition methods for their specific needs.

6 CONCLUSIONS AND FUTURE WORK

To facilitate an environment where multiple users can operate collaboratively in a multi-user CAx design session without data conflict, a new feature-based locking/reservation method has been developed. This feature-based locking method as well as the integration of collaborative feature constraints and collaborative user constraints successfully fulfills the research objective, *to develop a data consistency architecture and system to protect data and prevent data conflict in a collaborative CAx environment*, as stated in section 1.2.

6.1 Benefits

The benefits of this method are many. Firstly, this method provides an easy to use and understand interface for users in a collaborative and multi-user CAx design environment to eliminate update and delete data conflicts. The three-state selector to reserve/lock/release features is simple to use and will help users collaborate within shared collaborative design space. The visual awareness conveyed by the different coloring of features as they are locked or reserved will serve as an additional avoidance method for data conflicts, as it lets users know where in a CAx model other users are editing, and which features they are allowed to edit.

Secondly, the collaborative feature constraints developed for this thesis will help protect precisely designed models as the CAx data is shared with several different users. This would be important in the example of a CAD interface feature that was designed exactly to fit with another part. By being able to create collaborative feature constraints, users can preserve painstakingly designed interface features while allowing other collaborating designers to safely work on other features of the model without the risk of modifying sensitive geometry.

Third, the administrative interface and collaborative user constraints will help a CAx design team define the roles of individuals on their team and protect their data from being changed inadvertently or by unauthorized individuals when it is shared with others. The collaborative user constraints also allow flexibility for smaller or less constrained design teams to be able to work together freely without meticulous setup. It also allows flexibility to add or remove feature lock, and collaborative feature constraints when the user who created them is unavailable.

Overall, this data consistency method allows collaborative CAx software to be used more safely and more effectively because it protects the data created by the software, and controls how the data is used, who can modify it, and who has access. The method outlined in this thesis could be implemented in many different ways depending on the application, but the ideas will improve the usability and overall quality of collaborative CAx software, and ultimately prevent data conflict and protect data in a multi-user design session.

6.2 Drawbacks and Weaknesses

One drawback to using the method in this thesis to prevent data conflict and otherwise protect data is the overhead required to reserve and lock features while they are being edited. The process of locking/reserving features may get tiresome for experienced users who are accustomed to a single user CAx application, and could deter adoption of an implementation of the method. This method, however, does require less overhead setup time than would a spatial decomposition method such as that developed by Marshall [15]. Although there is some overhead to reserve or lock features, the protection to the data that this system provides will save users time in the long run as they will avoid having to deal with errors and data cleanup. This overhead could also be minimized with a well-designed and intuitive interface for the implementation.

Another weakness of this method is that it does not have a well-defined way to control the creation of new features in a CAx model. In this method the only way to prevent users from creating new features in a model is to completely disable the user's ability to create features. This is an inherent weakness of this method and in order to improve collaboration, more flexible methods of controlling the creation of new features in a model should be explored.

6.3 Future Work

While much has been done in this thesis to show its effectiveness, additional work could be done to expand this research. Further work includes:

- Integrating the method directly into a wider variety of CAx systems such as CAM, FEA, CAD drawings, and CAD assemblies, and analyzing the method's effectiveness in actual design situations.
- Expanding collaborative feature constraint definitions and looking at how collaborative feature constraints could work with other CAx systems.
- Performing further usability testing with different user interface configurations for the locking/reservation method, collaborative feature constraints, and administrative interface to further enhance the ease-of-use of implementations of this method.
- Investigating how feature reservation/locking could integrate with other CAx model decompositions, such as spatial decomposition, to create hybrid decomposition methods.

This investigation could improve control over the creation of new features in a CAx model.

- Exploring additional ways to promote better user awareness in a CAx model using labels, colors, etc.
- Optimization of bandwidth usage and network load using this method should be investigated. A server-push system such as that developed by Winn could be used [20].
- Investigating ways to improve extensibility to make future integrations and development of this system more scalable.

REFERENCES

[1] E. Red, V. Holyoak, C. G. Jensen, F. Marshall, J. Ryskamp, and Y. Xu, "ν-CAx : A Research Agenda for Collaborative Computer-Aided Applications," *Computer-Aided Design*, vol. 7, pp. 1–18, 2010.

[2] R. Bidarra, E. van den Berg, and W. F. Bronsvoort, "A Collaborative Feature Modeling System," *Journal of Computing and Information Science in Engineering*, vol. 2, no. 3, p. 192, 2002.

[3] S. Zhou, X; Li, J; He, F; Gao, "Web-based synchronized collaborative solid modeling system." Chinese Journal of Computer Integrated Manufacturing Systems 9, pp. 960–965, 2003.

[4] K. Ramani, A. Agrawal, M. Babu, and C. Hoffmann, "CADDAC: Multi-Client Collaborative Shape Design System with Server-based Geometry Kernel," *Journal of Computing and Information Science in Engineering*, vol. 3, no. 2, p. 170, 2003.

[5] L. Qiang, Y. F. Zhang, and a. Y. C. Nee, "A Distributive and Collaborative Concurrent Product Design System through the WWW/Internet," *The International Journal of Advanced Manufacturing Technology*, vol. 17, no. 5, pp. 315–322, Feb. 2001.

[6] U. von Lukas, "Collaborative Geometric Modelling using CORBA Services," *OOGP'97*, vol. 7, no. 2, p. 91, 1997.

[7] J. D. Ryskamp, "Developing Intelligent Engineering Collaboration Tools Through the use of Design Rationale," Brigham Young University, 2010.

[8] A. Agustina, F. Liu, S. Xia, H. Shen, and C. Sun, "CoMaya: incorporating advanced collaboration capabilities into 3d digital media design tools," in *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, 2008, pp. 5–8.

[9] "Codoxware: Connecting people and documents," 2011. [Online]. Available: http://www.codoxware.com/. [Accessed: 04-Jan-2012].

[10] R. Urbano, *Oracle Database Advance Replication*, vol. 33, no. Database issue. 2007.

[11] S. Jing, F. He, S. Han, X. Cai, and H. J. Liu, “A method for topological entity correspondence in a replicated collaborative CAD system,” *Computers in Industry*, vol. 60, no. 7, pp. 467–475, 2009.

[12] J. Bu, B. Jiang, and C. Chen, “Maintaining semantic consistency in real-time collaborative graphics editing systems,” *IJCSNS*, vol. 6, no. 4, p. 57, 2006.

[13] L. Chen, L; Song, Zhijie; Feng, “Internet-enabled real-time collaborative assembly modeling via an e-Assembly system: status and promise,” *Computer-Aided Design*, vol. 36, no. 9, pp. 835–847, Aug. 2004.

[14] K. Lin, D. Chen, C. Sun, and G. Dromey, “Maintaining constraints in collaborative graphic systems: the CoGSE approach,” in *ECSCW 2005*, 2005, no. September, pp. 185–204.

[15] F. Marshall, “Model Decomposition and Constraints To Parametrically Partition Design Space In a Collaborative CAx Environment,” *Constraints*, no. December, 2011.

[16] C. Cera, I. Braude, I. Comer, T. Kim, J. Han, and 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, p. 10.

[17] S. Chan, M. Wong, and V. Ng, “Collaborative solid modeling on the WWW,” *Proceedings of the 1999 ACM symposium on Applied computing - SAC ’99*, pp. 598–602, 1999.

[18] W. Li, “An Internet-enabled integrated system for co-design and concurrent engineering,” *Computers in Industry*, vol. 55, no. 1, pp. 87–103, Sep. 2004.

[19] Y. Xu, “A Flexible Context Architecture for a Multi-User GUI,” no. December, 2010.

[20] J. D. Winn, “Integration of Massive Multiplayer Online Role-playing Game Client-server Architectures with Collaborative Multi-user Engineering CAx Tools,” no. April, 2012.