Multi-User Methods for FEA Pre-Processing

Prasad Weerakoon
Brigham Young University - Provo

Multi-User Methods for FEA Pre-Processing

Prasad Weerakoon

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

June 2012

ABSTRACT

Multi-User Methods for FEA Pre-Processing

Prasad Weerakoon
Department of Mechanical Engineering, BYU
Master of Science

Collaboration in engineering product development leads to shorter product development times and better products. In product development, considerable time is spent preparing the CAD model or assembly for Finite Element Analysis (FEA). In general Computer-Aided Applications (CAx) such as FEA deter collaboration because they allow only a single user to check out and make changes to the model at a given time.

Though most of these software applications come with some collaborative tools, they are limited to simple tasks such as screen sharing and instant messaging. This thesis discusses methods to convert a current commercial FEA pre-processing program into a multi-user program, where multiple people are allowed to work on a single FEA model simultaneously.

This thesis discusses a method for creating a multi-user FEA pre-processor and a robust, stable multi-user FEA program with full functionality has been developed using CUBIT. A generalized method for creating a networking architecture for a multi-user FEA pre-processor is discussed and the chosen client-server architecture is demonstrated. Furthermore, a method for decomposing a model/assembly using geometry identification tags is discussed. A working prototype which consists of workspace management Graphical User Interfaces (GUI) is demonstrated.

A method for handling time-consuming tasks in an asynchronous multi-user environment is presented using Central Processing Unit (CPU) time as a time indicator. Due to architectural limitations of CUBIT, this is not demonstrated. Moreover, a method for handling undo sequences in a multi-user environment is discussed. Since commercial FEA pre-processors do not allow mesh related actions to be undone using an undo option, this undo handling method is not demonstrated.

Keywords: multi-user collaboration, collaborative design, multi-user decomposition, multi-user architectures, collaborative architectures, CAx, multi-user FEA pre-processing, CUBIT, CUBIT Connect, workspace assignment, workspace decomposition

ACKNOWLEDGEMENTS

I would like to thank all those who have helped me complete my research. I would like to thank my advisor, Dr. Ed Red, for accepting me as a graduate student, and also for guiding me every step of the way. I would like to acknowledge my committee members Dr. Greg Jensen and Dr. Steven Benzley for their support and advice throughout my research. I am also thankful to the industry members of the Center for e-Design and the National Science Foundation (NSF) for funding this research. I am grateful to Miriam Busch who has been a guiding light throughout my graduate career at Brigham Young University.

I would like to thank Dr. Karl Merkley and all the staff at CSIMSOFT for helping me with understanding CUBIT's source code. I would also like to thank Dr. Chia-Chi Teng, James Wu, Tim Bright, for their time help with getting CUBIT Connect implemented. I would like to acknowledge Jared Briggs, and Aaron Ford for proof reading my thesis and for their invaluable feedback.

I owe my deepest gratitude to my family; this thesis would not have been possible if it were not for my parents, Shantha and Sardha Weerakoon and my wife Dinali. Lastly, I offer my appreciation to all of those who have supported me and helped me in any respect during the completion of my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Current computer-aided engineering (CAx) applications allow only a single user to create/manipulate geometry, or prepare the model/assembly for analysis at a given time (Red, Jensen, et al. 2011). Though the technology is available to enable multiple users to interact with each other and make modifications in online environments within the same platform, these advancements have not yet made their way into the industry-wide engineering process. An example of this would be the advancements in massively multiplayer online role-playing games (MMORPG) (Wikipedia 2012) such as World of Warcraft, Everquest 2, etc. Even though it is not a simple task to convert traditional CAx tools to an environment similar to that of an MMORPG, most of the networking/programming principles behind the MMORPG platforms could be used as a basis for implementing a multi-user architecture for CAx applications.

In engineering, it is a basic requirement for products to be analyzed through a finite element analysis (FEA) program before starting full-scale production. FEA, undoubtedly, is one of the most time-consuming tasks in the product development (PD) phase. Mesh generation/cleanup is the most time-consuming process within FEA (Owen, et al. 2008). At present, when performing FEA pre-processing, only a single engineer can check-out the master model/assembly at a time and make the required modifications. That engineer then passes the model to another whom then works on a different part of it. For models/assemblies that have hundreds of millions of elements, this takes a substantial amount of time. This serial process is also true for Computer Aided Design (CAD) software as well (Lee, Kim and Banerjee 2010).

The Collaborative CAx methods developed by ν-CAx (new-CAx) researchers at Brigham Young University (Red, Holyoak, et al. 2010) would greatly benefit the field of multi-user FEA. The primary focus of the aforementioned research was CAD software (Red, Jensen, et al. 2011). Although FEA programs are similar in some ways to the CAD software, they have to be treated separately when developing multi-user environments.

1.1 **Motivation**

Having more than one person collaborate on a task can decrease the task completion time considerably. This is also applicable when it comes to using CAx tools to design and analyze a product. It can be safely argued that, when multiple engineers work on the same model simultaneously, with predetermined workspaces and boundaries, this level of collaboration would decrease the overall PD time significantly.

In 2005, a survey at Sandia National Labs was conducted to determine where the majority of time in Finite Element modeling and simulation, is being used (Owen, et al. 2008). It was found that 73% of the time was consumed in developing a solid model for analysis, including meshing, and applying boundary conditions, etc., and only 4% of the time was being utilized by the actual running of the simulation. Of this 73%, much of the time was spent cleaning up the model. A multi-user pre-processing approach such as CUBIT Connect, as introduced later, is an ideal solution to this problem. Allowing multiple engineers to collaboratively clean-up a mesh of a single solid model would reduce the pre-processing time significantly, as demonstrated later.

If multiple users are allowed to access and create pre-determined components of a product at the same time, as well as see the entire model on their screens, it can potentially decrease the modeling time and enhance the entire engineering change order system. Since engineers would be able to see each other's work on their screen, it would be easier for them to understand and see potential problem areas within the modeling stage itself.

This research involves using an existing FEA pre-processing software tool to develop the discussed multi-user architecture. The main purpose of this thesis is to conduct the research that will define and propose new architectures for multi-user FEA, develop methods of assigning workspaces, decomposing a model or assembly (Bu, Jiang and Chen 2006), and identifying overall conflicts with multi-user CAx implementations relating to FEA pre-processing.

1.2 Research Objectives

The finite element method (FEM) is a numerical method that transforms the partial differential equations of continuum mechanics into a large number of linear algebraic equations that is then solved using a computer (Zienkiewicz, Taylor and Zhu 2005). This is a recursive method that uses a lot of computing power and time (Balling n.d.). Therefore, this process is one of the main bottlenecks in PD. With the advancement of computers, a substantial amount of research has been done in finding ways of optimizing these FEM algorithms.

This thesis proposes a method to reduce the amount of time used when performing finite element analysis mesh generation by distributing the FEA portion of an engineering project amongst multiple engineers working simultaneously. This is done by allocating a specific area of the model to each engineer to work on. This is not feasible using current software as all mainstream commercial FEA pre-processors are created for serial FEA modeling rather than parallel (multi-user) collaborative product development, as mentioned in the problem statement.

The objective of this thesis is to convert an existing FEA mesh generation software program into a multi-user platform, and then study and propose ways of decomposing complicated CAD models to minimize or eliminate conflicts among collaborating engineers. An

FEA mesh generation program called CUBIT, developed by Sandia National Laboratories, will be used as the platform in which all testing will be done (Sandia National Laboratories n.d.). Using the data collected from testing the proposed multi-user FEA pre-processor software, a new standard for FEA model decomposition in a multi-user environment will be recommended.

The main objectives for this thesis are:

1. Modify an existing FEA program to allow multiple users (clients) to simultaneously edit and observe a common complex model or assembly.
2. Propose and test a candidate networking architecture for the multi-user FEA platform.
3. Develop methods to partition/decompose the model among a set of users and to regulate editing interactions of the distributed users. These methods may extend to geometric partitioning/decomposing the model-space among several users, with constraint surfaces that will limit model access to assigned users.
4. Develop methods to regulate the processing of time-constrained algorithms among multi-users in a collaborative FEA processing session.
5. Develop new methods for handling undo sequences when several users are collaborating on a model or assembly, and when merging modeling operations from several distributed users.

2 BACKGROUND

2.1 Current State of Multi-User FEA Applications

Collaboration using CAx applications is not a new idea. According to Nidamarthi (Nidamarthi, Allen and Sriram 2001) the first ideas in engineering collaboration appeared in the early 1990s. In 1991, Sriram predicted that engineering collaboration would be done on "a network of computers/users" where all users would share information through a centralized communication medium or a server (Sriram, Logcher and Fukuda 1991). Since then, there have been various advances in collaborative engineering using CAx applications as outlined below.

A considerable amount of research has been conducted in the field of multi-user CAD applications (Red, Jensen, et al. 2011); however, there has not been any notable research done in the field of multi-user FEA applications. For the past 3 years research directed by Dr. Greg Jensen and Dr. Edward Red (Red, Holyoak, et al. 2010) (Xu, Red and Jensen 2011) (Red, Jensen, et al. 2011) at Brigham Young University has yielded several multi-user CAD tools. NX Connect uses a client-server architecture and updates each client's model using an SQL database (Red, Holyoak, et al. 2010) . CATIA Connect uses a similar architecture and both NX Connect and CATIA Connect use native APIs to pass the changes to the server (Red, Jensen, et al. 2011).

Research has been done at Stanford University on developing a collaborative FEA program that can handle new technologies without relying on an individual software developer (Peng, et al. 2000). This program allows developers to submit and update the software collaboratively. There has also been research done on how to collaboratively design electromagnets on an FEA system by Anbo (Anbo, et al. 2002).

Furthermore, research has been conducted at both Beijing Jiaotong University and Texas Tech on making FEA collaboration more effective by integrating multiple CAx tools to FEA programs at various stages in the analysis phase of the PD cycle (Yu, et al. 2010). However, none of the aforementioned studies allows collaborative model modification at the same time. Google Docs (Google n.d.) is a well-known example of a real-time multi-user collaborative environment. Unfortunately, implementing a similar architecture in CAx applications may not be straightforward.

As demonstrated by Winn (Winn 2012) some elements of multiplayer games (MMOG- Massively Multi-Player Online Gaming) can be incorporated into multi-user CAx software to make networking and model management more efficient.

2.2 **Motivation**

FEA software is most commonly employed in new product development and/or product refinement, but is also often used in failure analysis and event reconstruction. FEA methodology is not exclusive to mechanical engineers. It is also widely used by civil engineers to analyze structures, buildings, bridges, etc., as well as by electrical engineers to analyze circuits and thermal casings and by chemical engineers to model chemical reactions and heat and mass transfer. Therefore, having multiple users work on the same FEA model/assembly at a will have a major impact on many areas of engineering.

File and model management methods in CAD have been proposed and demonstrated successfully. Also, a system to automatically decompose a complex structure into an assembly containing sub-parts has been demonstrated by two researchers in Lithuania (Bonneau and Gabrielaitis 2009).

Marshall, a researcher at BYU, has successfully demonstrated CAD workspace decomposition using Siemen's NX CAD software (Marshall 2011). Even though the FEA workspace is different from the CAD workspace, an approach similar to that of Marshall's can be employed in the FEA environment. The same partitioning methodologies (such as using coordinate planes) can be implemented in the FEA analysis space.

3 METHOD

3.1 Multi-User FEA Pre-Processor

The main goal of an FEA pre-processor is to prepare the CAD-generated model for finite element analysis. Preparation includes manipulating geometry (generating meshes, correcting meshes, etc.) to make the model suitable for FEA. This section discusses methods involved in converting a traditional single user FEA pre-processor into a multi-user environment.

3.1.1 Capturing User Interaction with the CAx Application

In a multi-user environment it is necessary for all users to receive model changes from the other users. This means a method to capture each user's model changes has to be developed. In NX Connect, the developers make use of undo marks to keep track of any model changes (Winn 2012) . Whenever an undo mark is created by a client's NX application, the changes that resulted in the undo mark are propagated to other users and their models are updated accordingly. This has been done at the API level. At the source level, a different approach should be utilized. Programming languages and styles may differ among CAx applications. However, when a user interacts with the GUI of a CAx program, there should be a common point (or points) where that interaction is captured and sent to the program's processing core for execution. Depending on the program's architecture, this can either be as simple as creating a command string (Merkley 2006) to be sent to the processing core, or as complicated as calling many different interdependent functions located throughout the source code. For example, CUBIT has a single point --CUBIT Interface--where all user-GUI interactions are sent to the core for processing.

Studying the source code and the architecture of the specific program is the best way to figure out the most effective method to capture all of the relevant data.

3.1.2 Filtering Relevant User Actions

In a multi-user environment, not all actions of a specific user need to be sent to other users. Only actions that are relevant to the overall model need to be shared among users (peer-relevant actions). Therefore, actions that are only relevant to the specific user and not relevant to the integrity of the overall model (user-specific actions) have to be filtered out.

Examples of user-specific actions are display actions (such as model rotations, panning, zooming, highlighting) and other actions such as opening/saving the model on your local computer, etc. Thus, a complete list of user-specific actions has to be compiled and those actions have to be filtered out when propagating actions to peer users.

3.1.3 Proposed Criteria for a Multi-User FEA Pre-Processor

After reviewing related literature and the architectures of some CAx programs, the following criteria for a multi-user FEA pre-processor were determined:

- Intercept all user interactions with the model
- Filter out user-specific actions (i.e. model rotations, pan, zoom, etc.)
- Propagate only the actions that are relevant to the overall model (peer-relevant actions) to other users
- Update the model on each client to reflect the changes from other users

Figure 3-1 shows the architecture diagram for the proposed multi-user FEA pre-processor.

With the implementation of the aforementioned approach, several critical issues arise:

- How to keep the models consistent across all client computers
- How to properly and securely manage incoming and outgoing data
- How to handle user workspaces
- How to keep users from interfering with other users' workspaces

Data management and workspace handling are discussed in detail in Chapter 3.

3.1.4 Synchronous Versus Asynchronous Architectures

In FEA mesh generation, some tasks take longer to execute than others. Therefore, a synchronous multi-user architecture may not be an ideal collaborative FEA pre-processing environment. For example, if User A receives a complex meshing command from User B that takes two hours to compute, executing that command on User A's computer would potentially freeze that computer making it difficult for User A to continue with his/her own work on the model. Therefore, the need to investigate an asynchronous multi-user FEA pre-processor architecture arises.

The characteristic of a synchronous architecture is that all users get updates from peers in real-time and those changes get reflected on their screens as soon as those peer-commands are received and executed on their machines. This means that the users have little control over the incoming commands from peers.

In an asynchronous architecture, all users get updates from peers in real-time. However, those users have the option of choosing which updates to execute on their computers, depending on their preferences. A time-considerate FEA command-handling architecture is discussed in section 3.4.

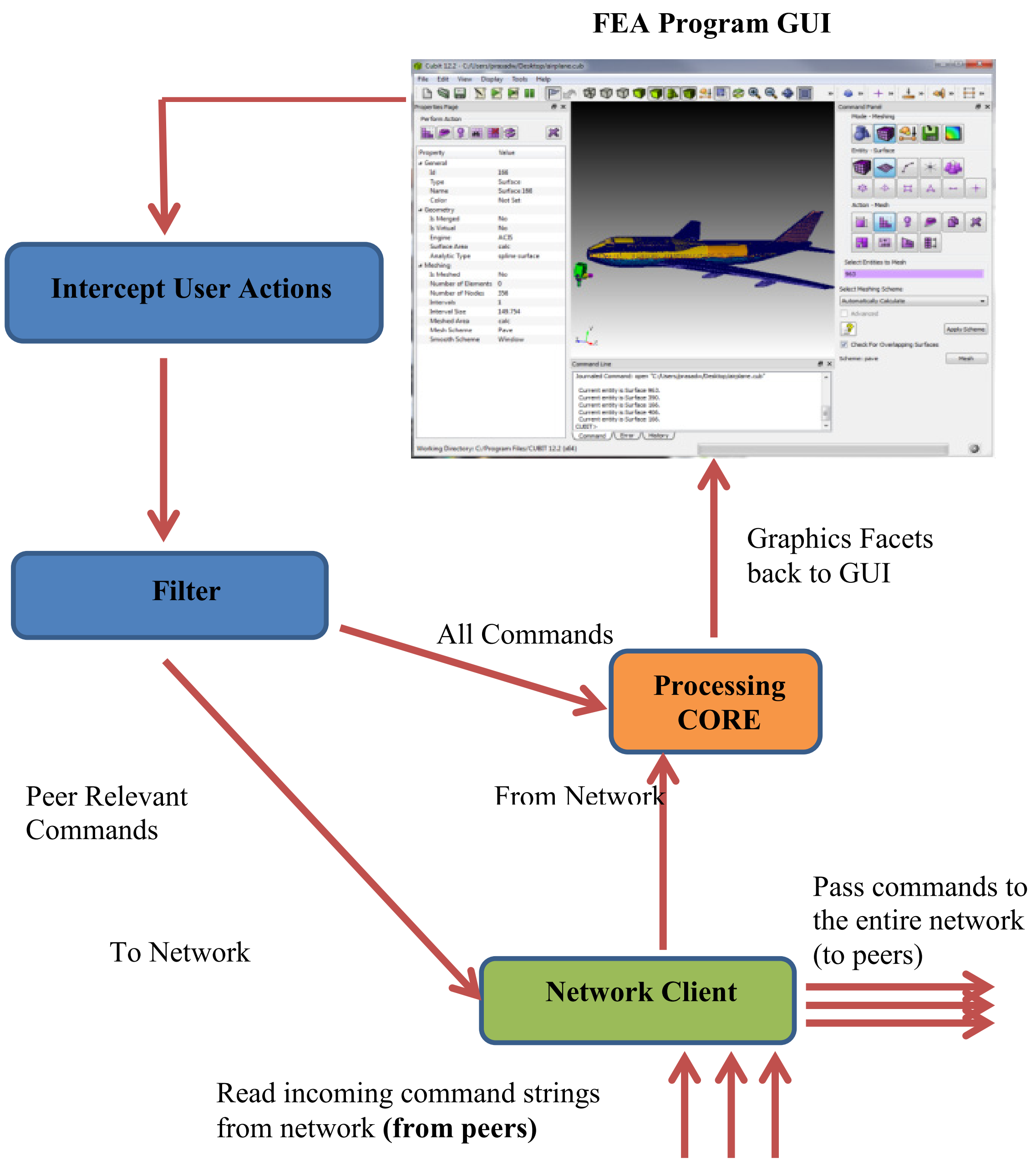

FEA Program GUI
Intercept User Actions
Filter
All Commands
Graphics Facets back to GUI
Processing CORE
Peer Relevant Commands
From Network
To Network
Pass commands to the entire network (to peers)
Network Client
Read incoming command strings from network **(from peers)**

Figure 3-1: Proposed Multi-User FEA Pre-Processor Architecture

3.1.5 Keeping Models Consistent Across Users

In the proposed implementation of the multi-user FEA pre-processor, each user's computer has a local copy of the model which gets modified. Changes from each user are sent to all users so their models are synced with each other. It is of utmost importance to have these models stay consistent between users. The whole purpose of a multi-user CAx application falls apart if users have different models.

Each CAx system has some sort of entity identification system to identify various entities that are created, such as volumes, faces, nodes, etc. Whenever additional commands are executed upon these geometries, they will identify the feature by the integer ID. When users receive peer commands, each user has to receive those commands in the same order as they were submitted in order to keep the integer ID system consistent. Figure 3-2 shows an example of when users have the same model, but different integer ID numbers. In this case, any further action executed upon these models will have different results. Therefore, by keeping these integer IDs uniform across all users, their models stay consistent. This will happen only if each user receives all changes from peers.

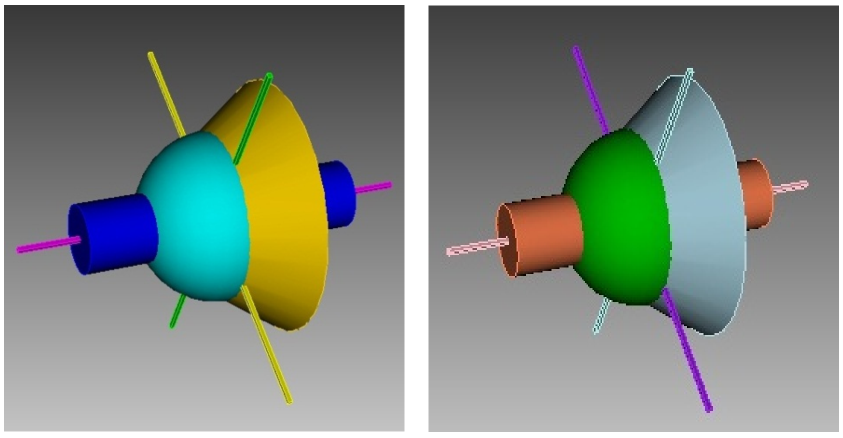

(a) (b)

Figure 3-2 a, b: Integer ID Inconsistency Issue

Figure 3-2 (a) & (b) shows two users in a multi-user environment creating multiple volumes that lead to the same overall model. Notice the difference in colors between the two models. This particular CAx application has a color system that denotes different volume numbers. Therefore, the difference in colors signifies a difference in volume ID tags even though the entire model was created in the same multi-user session and both models are geometrically identical. Any further action on these two models will result in different outcomes for the two users. For example: if the user on the left hand side issues a command to mesh volume 1 (the dark blue cylinder) and it is sent to the user on the right hand side, volume 1 may not be the same cylinder for that user. It may result in that user getting a mesh on some other geometry which, on their computer, is identified as volume 2.

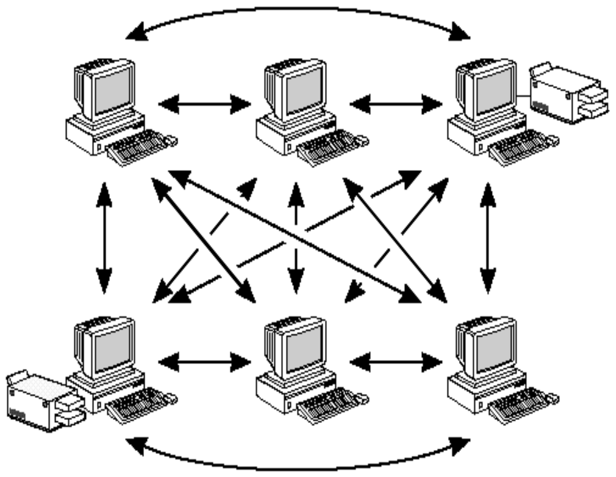
To avert this problem, all users' commands have to be sequentially executed on each user's computer. This will involve routing commands to a central queue on the network and then sending those commands back to all users for execution in the same order.

3.2 **Networking Model for Multi-User FEA Pre-Processor**

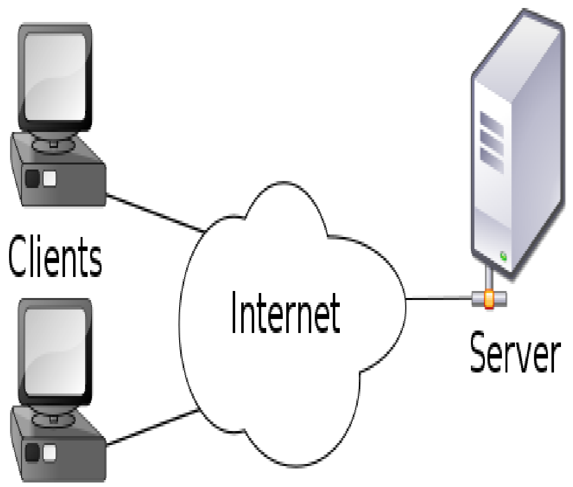
For a multi-user CAx application, another important aspect is to have a reliable networking architecture that can handle multi-user connections reliably and efficiently. A slow, unreliable connection would defeat the purpose of a multi-user environment and it can also be counterproductive.

3.2.1 Centralized Server

As mentioned in Section 3.1, it is important to properly and securely manage the incoming and outgoing data streams of all users in the collaborative session. Several networking architectures were considered in order to select the most suitable one for a multi-user environment. A peer-to-peer (P2P) model and a client-server (CS) model were investigated (sections B.3, B5). Figure 3-3 shows network diagrams of each of these models.



(a) P2P Networking Model
http://picasaweb.google.com/lh/photo/WIN006CImw1DeqqCcuWeog



(b) CS Networking Model
http://en.wikipedia.org/wiki/File:Client-server-model.svg

Figure 3-3 a, b: Schematics of P2P and CS Networking Models

Chapter 4 discusses these implementations and their advantages and limitations in a multi-user environment. A centralized system for handling all connections is proposed and implemented in this research instead of the peer to peer system. The centralized system has a server that is connected to all clients, enabling the server to act as the central hub of information-sharing. The determined functionality criteria of the server are:

- Be able to identify commands from individual users, as well as direct data to individual users

- Store a master log file of all incoming commands that can be used to recreate the model, if needed
- Store a log file containing all commands from each user
- Be able to handle users who connect at different times and send updates to those users to get their models synchronized to the latest version (asynchronous collaboration)

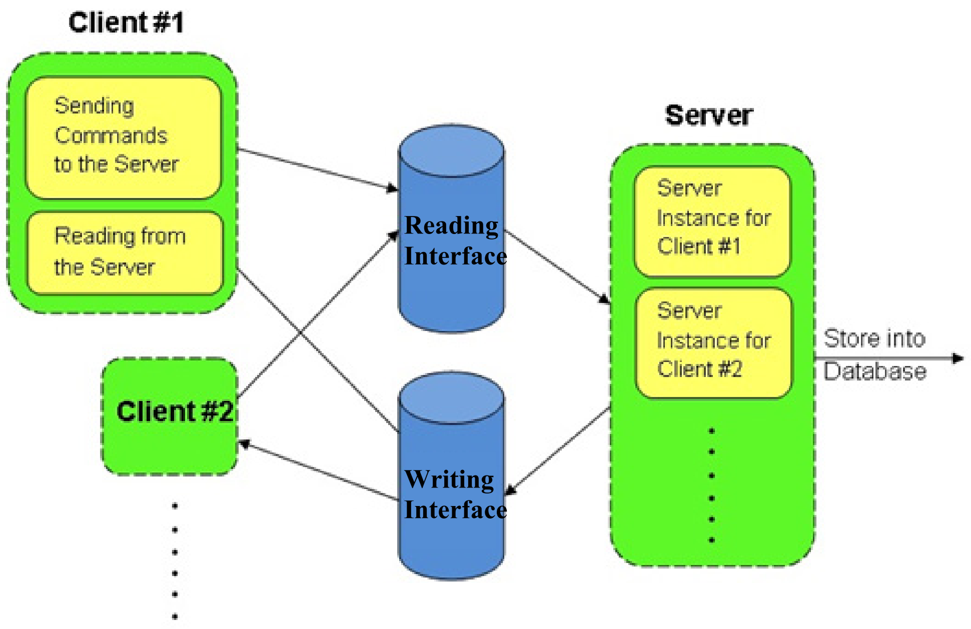


Figure 3-4: Proposed Networking Architecture for Multi-User FEA Pre-Processor

3.2.2 Data Capture Module on a Client Computer

The data capture model resides on the client computer and has two main responsibilities:

1. Gather changes from the client's CAx application and send those to the server
2. Receive changes from the server and send those to the client's CAx application

For the purposes above, a multi-threaded data capture module is proposed. One thread will be dedicated to gathering and sending local data while the other will be dedicated to receiving updates from the server and sending those off to the CAx application. The determined functionality criteria of the client data capture module are:

- Establish a stable connection with the server
- Communicate with the CAx application and gather all peer-relevant actions
- Send these commands to the server along with the IP address of the client computer
- Communicate with the server and gather all data coming in from other users
- Pass those commands to the CAx application for processing

3.2.3 Multi-User Tagging

To identify different users in a multi-user environment, a unique identifier can be used to distinguish a user from all other users. A unique identifier can be obtained by using a combination of:

- A unique username for each user
- IP address of the user's computer

The username should be coupled with the IP address of the user's current computer for security purposes. Since IP addresses are computer specific, it should not be used as the only identifier. These identifiers can then be encoded to the message passed with the user's action to the central server. The server can then propagate the command, along with the user ID, to other multi-users in the multi-user session. Each client computer can decode the user ID from the data coming from the server. A new system of GUIs and methods to handle users and security in a multi-user environment should be established. These GUIs and security measures are discussed and demonstrated in Section 4.4.2.

Using this unique user ID, the following can be implemented in a multi-user environment:

- Workspace assignment
- Easier moderation/administration
- User’s view angle observation can be implemented (if this data is sent as a peer-relevant command)
- Assign a unique identifier on to the mouse curser of each user when viewing the whole model with user interaction
- More security measures can be implemented
- A peer to peer video/chat messaging system can be implemented

Workspace Assignment using User ID

The unique ID allows you to assign regions in which an individual user is authorized to work. Each client computer can pull workspace assignments from a central database using the user ID. This is discussed further in Sections 3.3 and 4.4.

Easier Moderation/Administration

An administrator can allow or disallow access to specific parts of a model/assembly to users via their user ID’s. This can be useful when giving access privileges in a multi-user system.

Note that this includes, but is not limited to workspace assignment, assigning user rights/roles (administrator, observer), etc.

User's View Angle Observation

By having a unique user ID, graphics data from a specific user (along with their user ID) can be tunneled to another user. The graphics facets can then be decoded by other users according to the accompanying user ID.

Assigning a Unique Identifier to the Mouse Curser of Multi-Users

Being able to see what multi-users are doing in a multi-user environment is something that is valuable to an administrator. It can also be used to study and enhance collaborative skills in the multi-user environment. The mouse curser, or other feature selection technique, can accompany the user's username and can be color coded according to username (Figure 3-5).

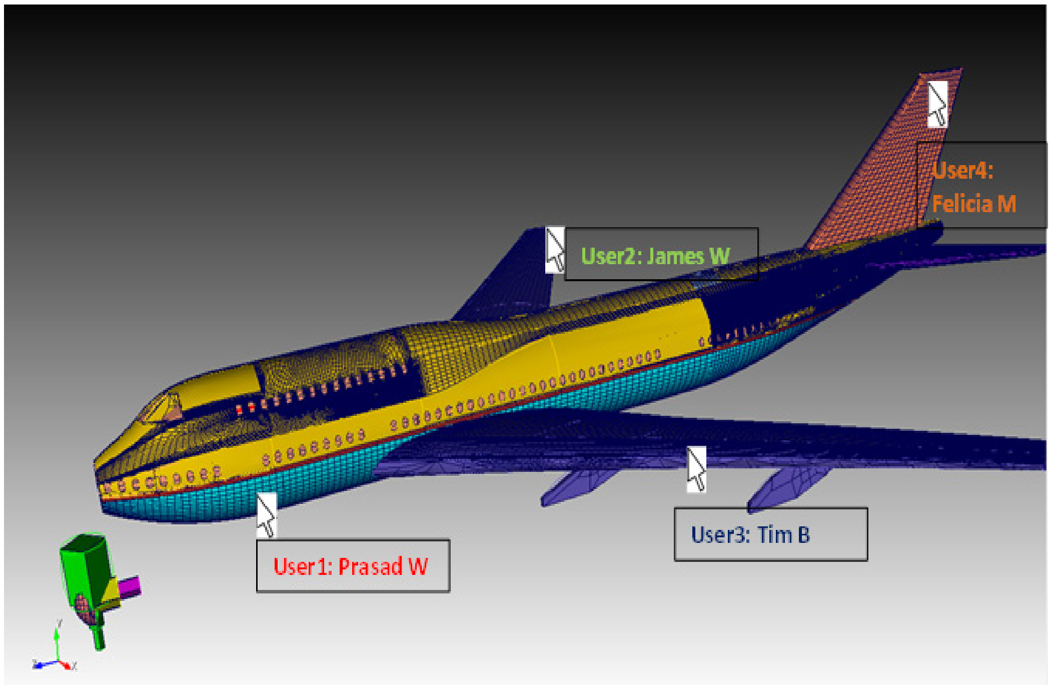


Figure 3-5: Mouse Cursor User Identification in a Multi-User Environment

Enhanced Security Measures

User IDs can be used to better control the multi-user environment by prompting for passwords along with the username or similar security authentication methods.

Peer to Peer Text/Video Messaging System

The User ID can be used to build a text, VOIP, or a video messaging system as demonstrated by Xu (Xu, A Flexible Context Architecture for a Multi-User GUI (Master's Thesis) 2010).

3.3 Model Decomposition/ Workspace Allotment

When a geometric model is ready to be analyzed through an FEA solver program (structural, thermal, CFD, etc.), the model is usually fully developed and no further complicated modeling needs to be performed. Unlike CAD (Marshall 2011), where assigning a workspace is somewhat ambiguous, FEA can be somewhat less challenging. As in other CAx applications, FEA workspaces can be divided using volumetric bounding planes. Since there are finite geometries in an imported model, FEA workspaces can be decomposed using these different geometries.

Typically, CAx systems use some form of geometry identification scheme to identify geometric identities. For example, two curves join to create a surface. Three surfaces join to create a volume, and two volumes combine to create a body. These identification schemes are application specific, but each CAx application has a well-defined scheme of geometry identification. Figure 3-6 shows a sample geometry identification system.

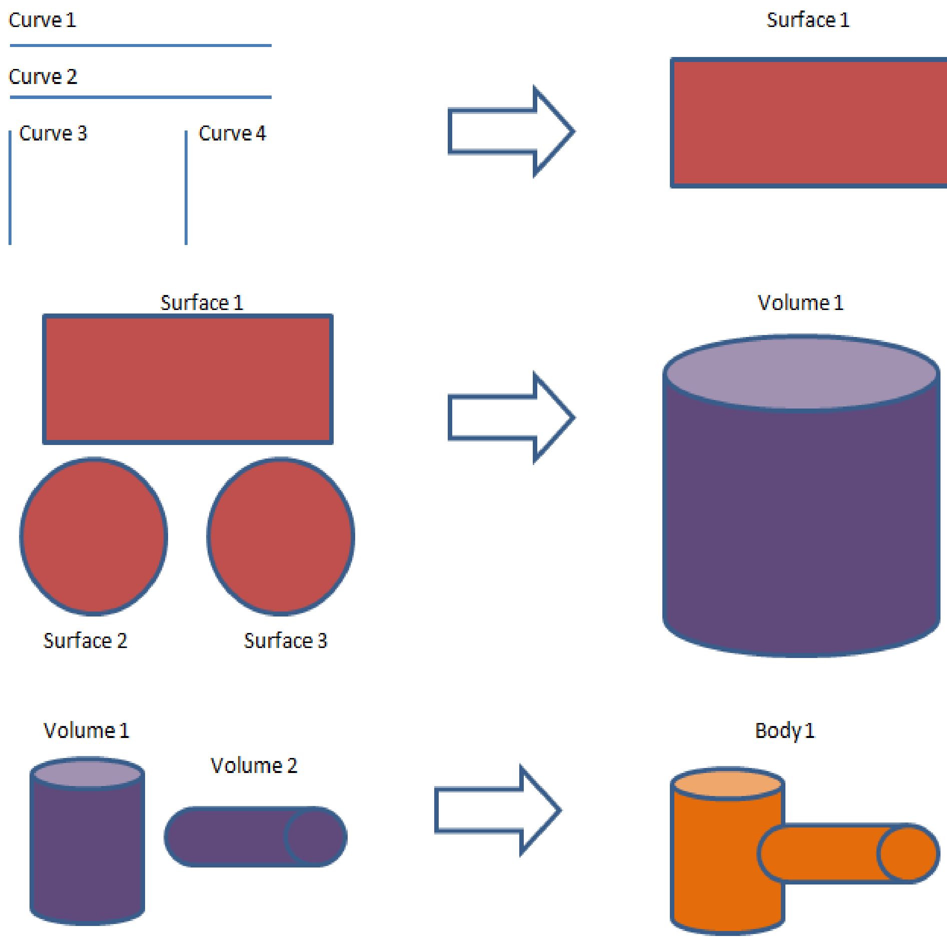
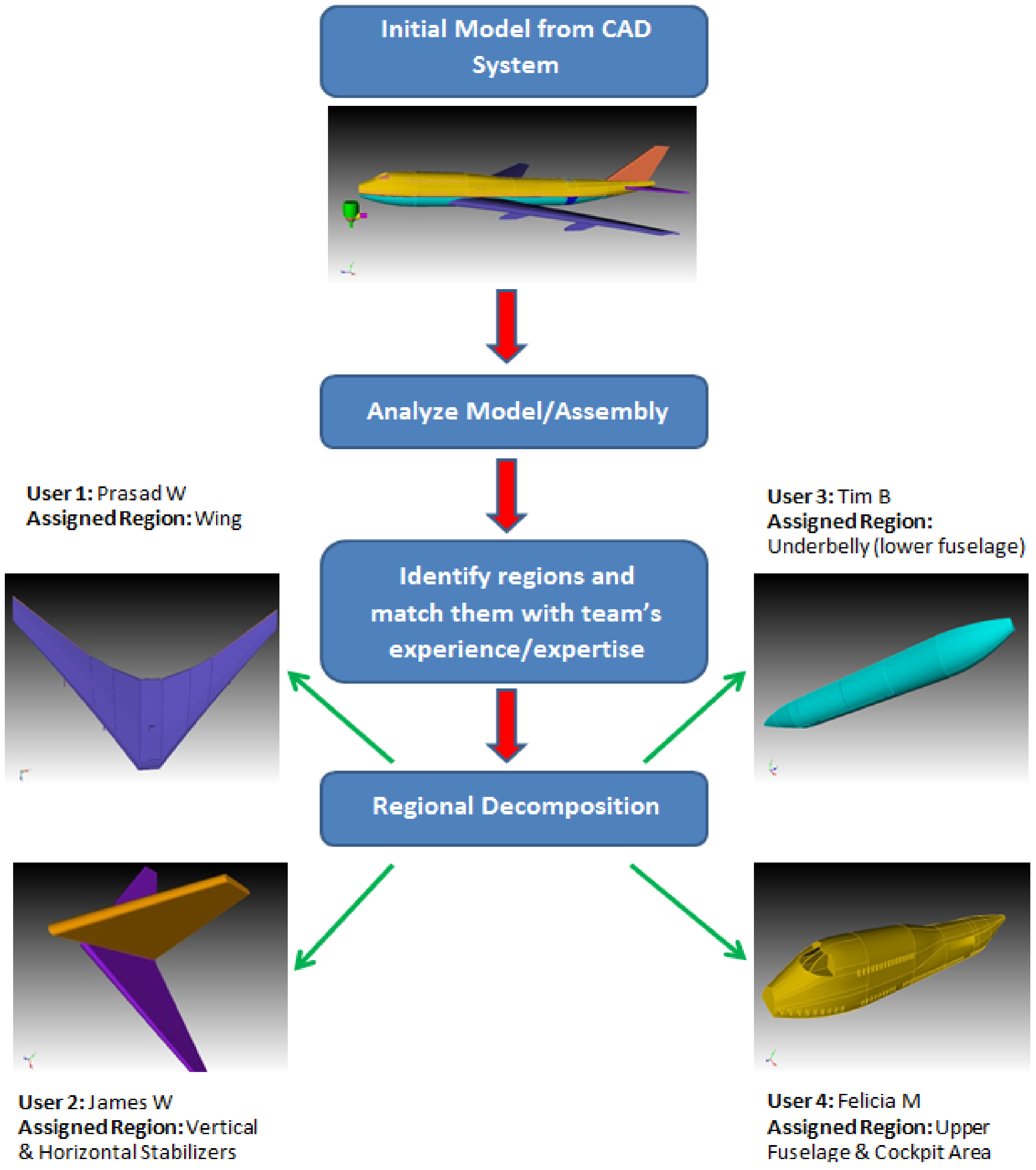


Figure 3-6: An Example Geometry Identification System

3.3.1 Assigning Workspaces Using Geometry IDs

As mentioned earlier, this identification scheme can be used to decompose a part or assembly and then assign workspaces to different users. Expertise varies among members of an engineering team. Therefore, a method to divide a model among users based on their areas of expertise is proposed. This method would allow a streamlined way to mesh and eliminate discontinuities in an FEA model within a multi-user environment. If you take a team meshing a race car as an example, some members would be experienced in meshing cylindrical regions such as wheels, while others may have expertise in meshing wings, or the cockpit area, nose cone, etc. Since each of these regions (cockpit, wheels, wings, etc.) is identified by their assigned geometry ID's in the FEA program, the model can be decomposed using those ID's.



Initial Model from CAD System
Analyze Model/Assembly
Identify regions and match them with team's experience/expertise
Regional Decomposition
User 1: Prasad W
Assigned Region: Wing
User 2: James W
Assigned Region: Vertical & Horizontal Stabilizers
User 3: Tim B
Assigned Region: Underbelly (lower fuselage)
User 4: Felicia M
Assigned Region: Upper Fuselage & Cockpit Area

Figure 3-7: FE Model Decomposition Scheme

Figure 3-7 shows an example method of FE model multi-user decomposition, where these steps would decompose the project and assign tasks like the following to a set of multi-users:

1) Import the model from the CAD system into the FE program

2) Team leader (administrator) inspects the model

3) Team leader identifies regions in the model and matches them with the team's expertise

4) Regions are assigned to team members using the geometry ID tags

3.3.2 Access Rights Using Geometry ID Tags

As illustrated in the previous section, a model or assembly can be decomposed using geometry tags. These tags can also be used to assign user workspaces and restrict them from modifying the workspaces of others. This method allows users to view other multi-user regions, but they cannot select or modify the geometry. The workspace essentially locks the user to a certain range of geometry IDs.

Once the region is assigned, the GUI enables mouse cursor interactions only to those regions assigned to a multi-user. The user can select, click, mesh or do any operation within that region. As mentioned, however, locking does not restrict the user from browsing other regions of the model or assembly.

3.4 Time-Consuming Task Handling

Some modeling/analysis tasks can take a considerable amount of time to complete, depending on the complexity of the task. In multi-user applications, if User A receives an action from User B that takes a long time to execute, it would potentially freeze the computer of User A until that task is complete. It is important distinguish between meshing algorithms (pre-processor) and solving algorithms (processor). Solving algorithms are executed once the FEA model is complete. Since running solving algorithms is an automated process, multi-user technology may not improve this. Especially in the case of FEM meshing algorithms, it sometimes takes days to complete a single task. That means User A has to stop whatever he/she was doing and sit idle. This happens in a synchronous multi-user environment where all users get updates from others in real-time without any prejudice. However, if each user was able to decide which updates he/she wants to accept from a list of incoming peer user updates, they can decide to let those complete whenever it is convenient for them. This can be addressed by an asynchronous multi-user architecture.

3.4.1 Using a Timer

A timer can be used to calculate how much time was spent on running a particular task. CPU time is the best method to use in such a situation. CPU time calculates how much time the CPU spent processing a task from a certain application. This time is different from the elapsed real time. CPU time is independent of the number of processing cores or threads a particular computer has. It is calculated by multiplying the total time the computer spent processing a task by the number of cores and by the number of threads per core. However, this time is dependent on the processing power of a single thread of a single core on that computer. This changes from computer to computer. However, a general idea of the time spent can be obtained from this number.

In the multi-user environment, a timer can be used to calculate the total CPU time the application utilized to run a certain algorithm. It can be achieved by sending a timer start command along with every user action that is sent to the CPU. This timer can be set to stop when the task completion notification is sent back to the GUI (Figure 3-8).

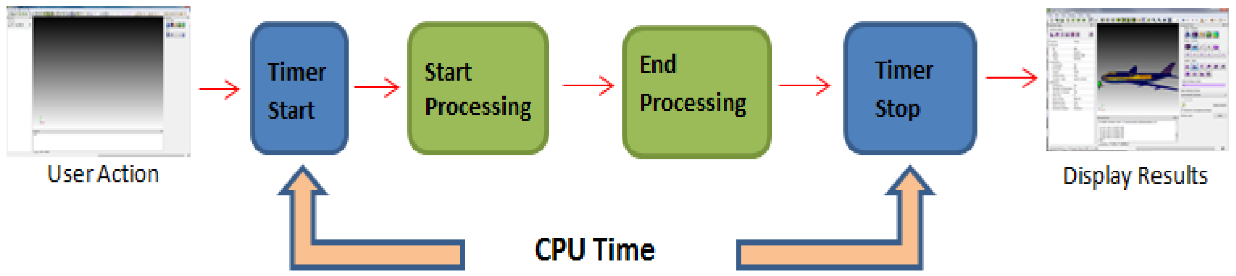


Figure 3-8: Calculating CPU Time in a Multi-User CAx Application

After obtaining the task completion time, it can be sent to the server along with multi-user command. This can be done similar to the method outlined in Multi-User Tagging (Table 1).

Table 1: User Commands Tagged with Command Execution Time (T)

Command	User ID	Utilized CPU Time (T)
cmd 1	user 1	*27.84600 seconds*
cmd 2	user 2	*5234.9600 seconds*
cmd 3	user 1	*145.5400 seconds*
cmd 4	user 3	*3.4600 seconds*
cmd n	user m	*y seconds*

The following flow diagram (Figure 3-9) shows the proposed method for handling time-consuming commands in an asynchronous multi-user environment. Commands from peers are analyzed one by one to see if they qualify for a pre-set “user preferred time”. This time is a variable that is changed by the user. If the peer-command’s execution time is less than the user preferred time, it is sent to the “Immediate Queue (IQ)”. The commands in the IQ are executed automatically. It should be noted that most of the current commercial FEA pre-processors are not multi-threaded; the method outlined here assumes a single -hreaded software architecture. When operations (GUI, meshing, etc.) are threaded and allowed to be run in parallel, time-consuming task handling in an asynchronous multi-user FEA environment does not cause a serious problem. That means users can continue to use the GUI while operations are executed in the background. Therefore, it is suggested that multi-user FEA pre-processing programs be multi-threaded to avoid the issue of the GUI freezing when time-consuming algorithms are run.

If the peer-command execution time is greater than the user preferred time, that command will be sent to the “Later Queue (LQ)”. The user ID from which this command originated is then flagged. Any further commands originating from the user ID are then sent to the LQ, regardless of the execution time. This is done to resolve conflicts that may originate from commands that build on top of commands that were sent earlier to the LQ. Once again, it should be noted that the method discussed here assumes that the multi-user FEA program is single-threaded.

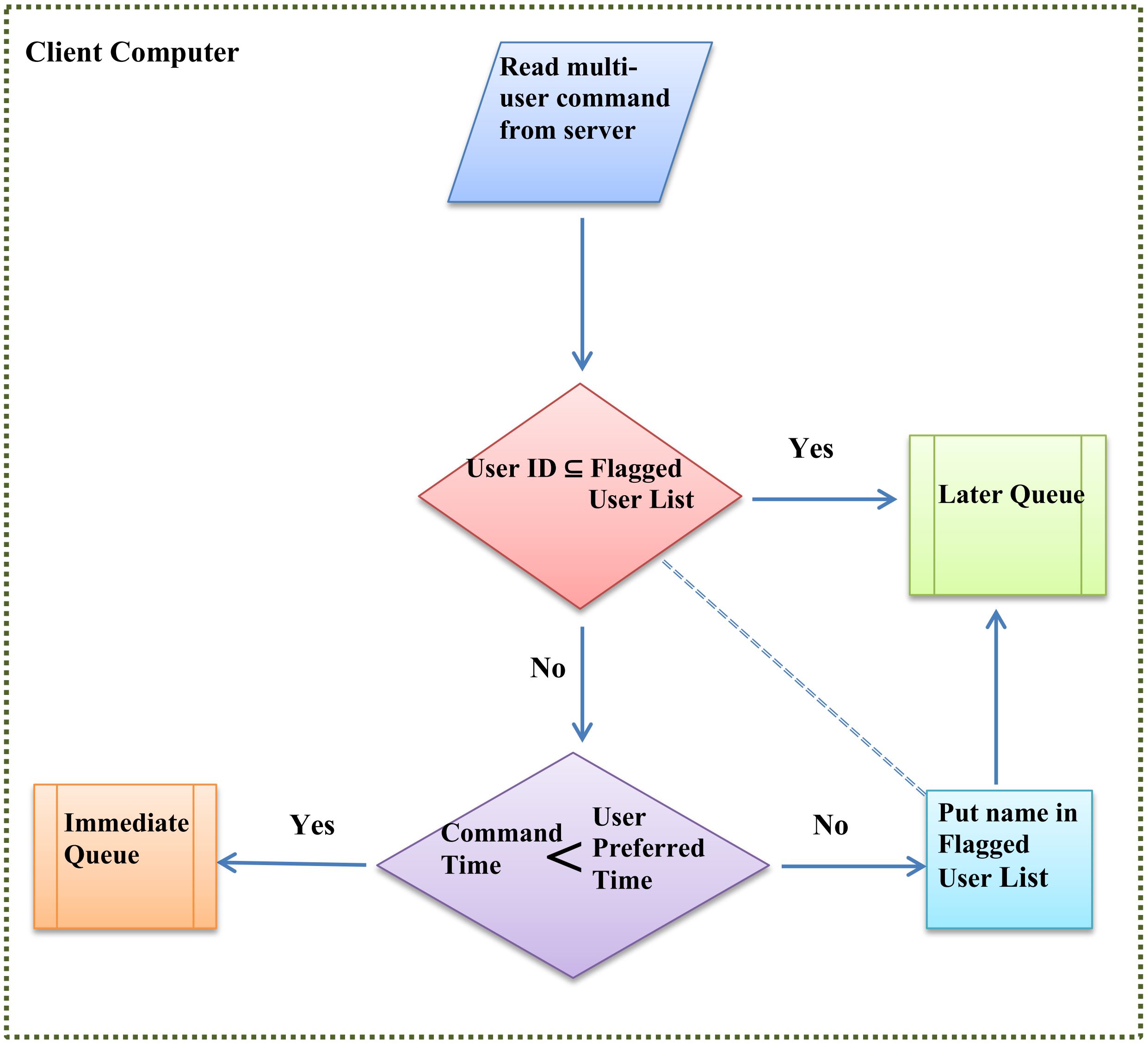
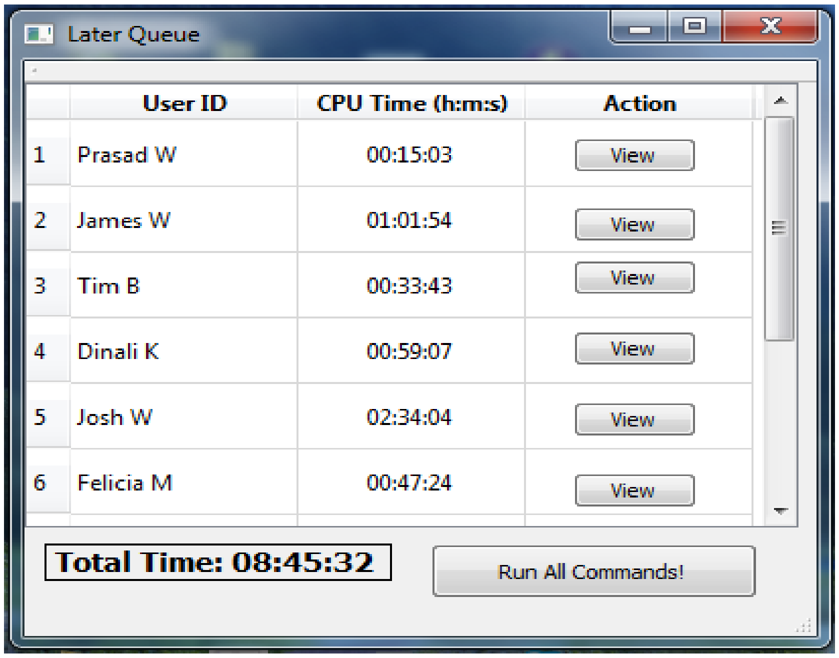

Client Computer
Read multi-user command from server
User ID ⊆ Flagged User List
Yes
Later Queue
No
Command Time < User Preferred Time
Yes
Immediate Queue
No
Put name in Flagged User List

Figure 3-9: Time-Consuming Task Handing Flow-Diagram

However, complications arise when multiple users are working on the same area of a model. Even though there may be clearly restricted workspaces, sometimes a different user might have to work on an area that another user previously worked on. If the previous user's commands are in the LQ, the new user's work might not get executed on another client's computer. This is because the previous user's work has not been executed yet. Therefore, it is absolutely necessary for all multi-users to be informed of such situations so that they can get their respective models up-to-date.

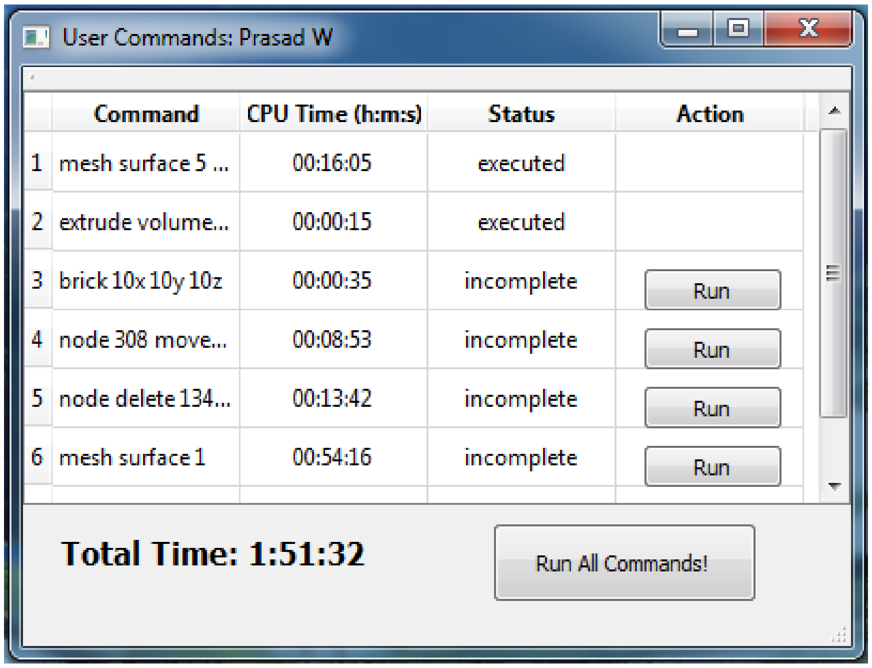
In this asynchronous multi-user environment, the user has the ability to run all commands that are in the LQ at their convenience (Figure 3-10). They can be given the choice of running all commands from specific users that are in the LQ (Figure 3-11).



	User ID	CPU Time (h:m:s)	Action
1	Prasad W	00:15:03	View
2	James W	01:01:54	View
3	Tim B	00:33:43	View
4	Dinali K	00:59:07	View
5	Josh W	02:34:04	View
6	Felicia M	00:47:24	View

Total Time: 08:45:32

Figure 3-10: LQ with a List of All Users


User Commands: Prasad W

	Command	CPU Time (h:m:s)	Status	Action
1	mesh surface 5 ...	00:16:05	executed	
2	extrude volume...	00:00:15	executed	
3	brick 10x 10y 10z	00:00:35	incomplete	Run
4	node 308 move...	00:08:53	incomplete	Run
5	node delete 134...	00:13:42	incomplete	Run
6	mesh surface 1	00:54:16	incomplete	Run

Total Time: 1:51:32

Run All Commands!

Figure 3-11: User Queue with a List of all Commands from One User

3.5 Multi-User FEA Undo Handling

After studying some commercial FEA pre-processing and post processing software, it was found that almost all FEA programs do not have an undo function (or Ctrl+Z) for most meshing-related operations. The software that were considered were; Altair's HyperMesh, Sandia Labs' CUBIT, and ANSYS.

HyperMesh allows undo in display operations and some auto-mesh operations, but does not offer undo in other related operations. However, HyperMesh has a reject function which can act as an undo function. The user is given the option to accept or reject each action. Unfortunately, after the user accepts the change, the user can no longer reject or undo it. The only option is to delete that change. ANSYS too has no undo option. Once a component is created in ANSYS it has to be deleted, or once a component is deleted it has to be recreated

(Haghichi and Nyquist 2007). CUBIT's undo capability is implemented for geometry commands such as geometry creation, transformations and Booleans. The undo functionality is not currently enabled for most meshing commands (Cubit 13.1 User Documentation n.d.). With these architectural limitations in mind, the following method is proposed for multi-user undo handling. Implementation of this method may not be possible due to the aforementioned architectural challenges.

Each command generated and sent to the server for propagation is appended with a user ID by the client computer. When the server receives the command, the message is again appended with a time stamp by the server. The server needs to have two distinct databases in order to implement this (Figure 3-12):

- Master Database (MDB): contains all commands from all users
- User Database (UDB): each user has a separate database that contains their commands

An Undo command is usually a reverse command that is meant to nullify the previous action. Undo only works for the current state of an application and the undo stack is cleared once the user exists out of it. However, in this implementation undo can be set up to reverse actions that were done in previous sessions as well.

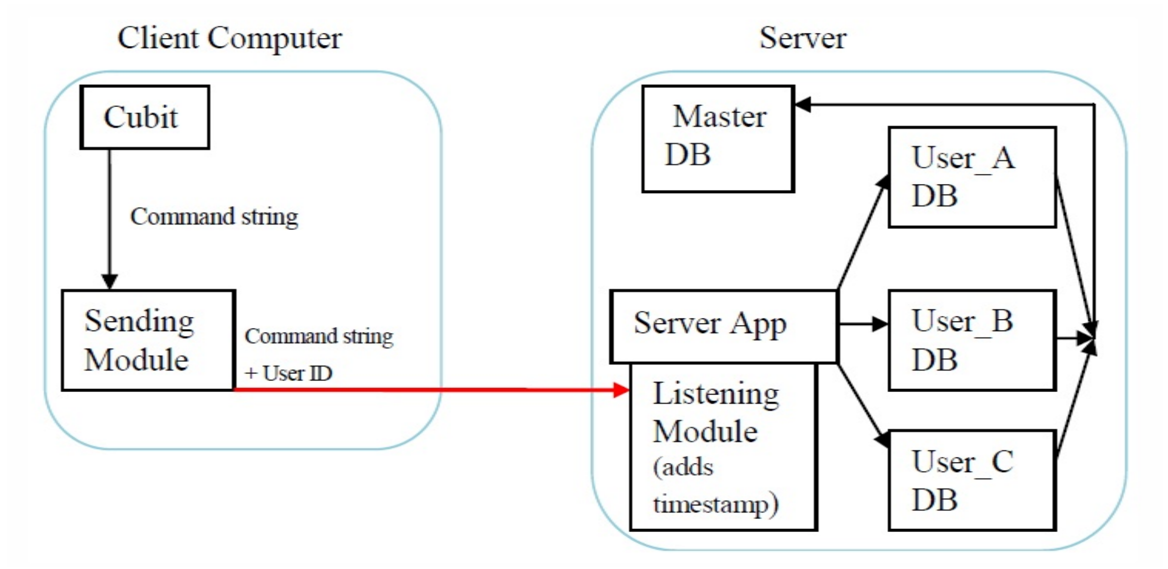


Figure 3-12: Undo Handling Architecture

When a user hits undo, the client computer looks up the last action of that particular user and issues a command that reverses that action. That command can then be propagated to other users in order to make the necessary changes on their computers as well. If one user's undo affects the work done by another user, then it should pop up a warning before execution. This can be done by checking if the undo command modifies any entities in a different user's workspace, thereby throwing an exception or a warning on the screen of the user issuing the undo if a conflict is found.

4 IMPLEMENTATION

4.1 CUBIT

CUBIT is a mesh generation tool developed primarily by Sandia National Laboratories and it is the primary platform on which mesh generation and geometry preparation (FEA pre-processing) are done at Sandia and its collaborating laboratories around the United States. Sandia has allowed CUBIT to be used for academic research purposes by providing its source code to academic institutions. Therefore, the ν-CAx research group at Brigham Young University (ν-CAx 2011) was able to obtain the CUBIT source code in order to develop multi-user CAx applications. It should be noted that CUBIT's source code is not open-source and is owned by Sandia National Laboratories.

4.1.1 CUBIT Architecture

The initial version of CUBIT was only a command-line mesh generation program. In 2003 a GUI, named CLARO, was added on top of the command-line version to make CUBIT more user-friendly (Merkley 2006). The original command-driven system and the command language were preserved in this new GUI driven CUBIT through an interface called "CUBIT-Interface". Whenever the user makes any modification to the model, the GUI creates a command string and passes it down to the CUBIT core through CUBIT-Interface. The core then runs the respective computational algorithms and updates the model while sending the new graphical facets to update the GUI (Figure 4-1).

The CUBIT core is written in C++, while Python and Qt Creator (Nokia Corporation 2012) were used to create the GUI. Since both high level and low level programming languages are used to program CUBIT, the management of data transfer between them is managed by a tool called SWIG (SWIG 2012).

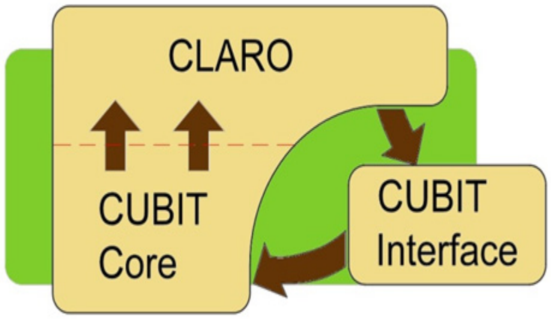


Figure 4-1: CUBIT-Interface

Note that CUBIT Interface is only used to pass the command strings down to the CUBIT core and only task completion events (flags) are passed back through it. Graphics are not passed back through CUBIT Interface. After running the relevant calculations, the core passes back graphics and other relevant data, to the GUI, through several other routes that are scattered throughout the source code.

4.1.2 Why CUBIT?

CUBIT is used as the primary development platform for multi-user FEA pre-processing because of the working relationship BYU has with Sandia National Laboratories and Computational Simulation Software, LLC (CSIMSOFT). CSIMSOFT is involved in selling commercial licenses of CUBIT. As mentioned earlier, the ν-CAx research group was able to obtain the source code of CUBIT, and therefore was able to make the necessary changes at the source level. The other multi-user CAD prototypes (NX and CATIA) developed at BYU use the respective programs' Application Programming Interface (API) calls to perform the multi-user tasks. Their functionality is dramatically reduced due to the limited availability of API libraries from the software manufacturers (Red, Jensen, et al. 2011). However, with the availability of the CUBIT source code, CUBIT Connect (the multi-user version of CUBIT) can therefore be programmed to be more robust and to have more functionality.

4.2 Multi-User FEA Pre-Processor: CUBIT Connect

CUBIT Connect is the name given to the multi-user version of CUBIT. The multi-user version has been created using the method outlined in section 3.1. CUBIT Interface (figure 4-1) provides an interception point for capturing all the changes that are made to the model by the user. Figure 4-2 shows the architecture of CUBIT Connect on the client side of the computer.

When the user interacts with the GUI, the GUI generates a command string that is then sent to the core for processing. This command string is sent through a function named "cmd()" in a C++ file "CubitInterface.cpp" in the source code of CUBIT. The unaltered cmd function is found in Appendix A.1 and the modified cmd function for multi-user handling is found in Appendix A.2.

After intercepting the command string, it is checked to see if it is a user-specific command. The user-specific commands are blocked from being sent to other users as mentioned in section 3.1. Some of the user-specific commands CUBIT uses are: graphics, draw, list, display, quality, preview, etc.). If it is a peer-relevant command, it is then sent to a server which broadcasts those messages to other users in the session.

4.2.1 Cubit Connect Prototypes – Client's Side

Two major prototypes were developed to demonstrate the multi-user environment using CUBIT. The first prototype consisted of a peer-to-peer architecture and the second architecture was a client-server (CS) one. The networking architecture is discussed in detail in section 4.3. Figure 4-2 shows the complete architecture on the client's side of the P2P implementation. The complete implementation details of the P2P program can be found in Appendix B.

Even though the basic concept of intercepting and transmitting user commands is the same for both implementations, there are some important differences between these. In the P2P implementation all commands are sent to that user's core for processing while sending the peer-relevant commands to other users. This means that the user's actions are implemented immediately, while commands from peer users are sent to a queue. The user is given the option to execute those peer commands at their convenience. This asynchronous architecture provides users to work on their portion of the model without interruptions caused by other users' updates. They can sync the model with other users' updates once they are done. However, this implementation has an important limitation. As discussed in section 3.1.5, keeping the model consistent between users is of utmost importance in a multi-user environment. In this asynchronous architecture, without a central model database, the model from user to user does not stay consistent. This is further discussed below.

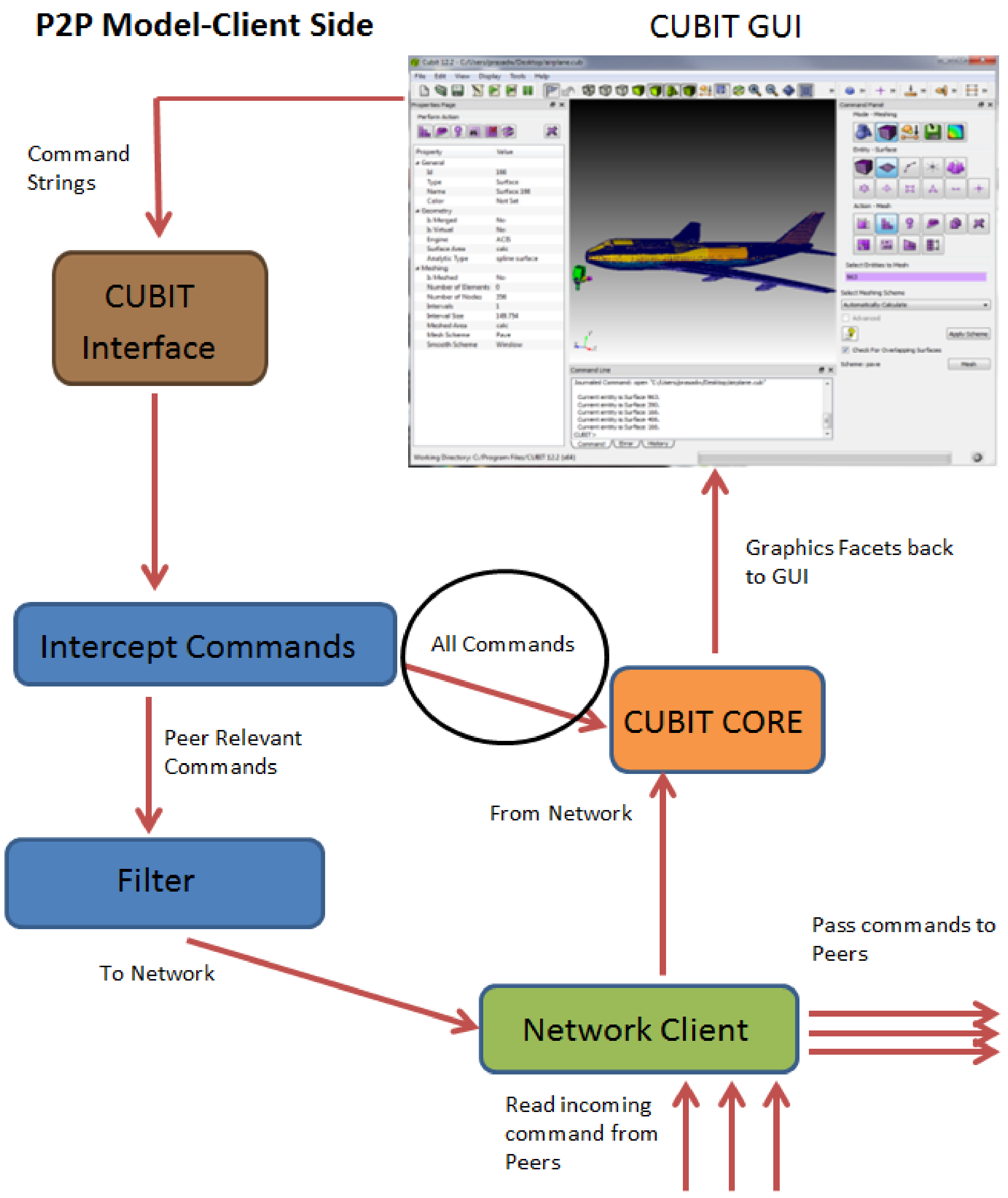


Figure 4-2 Client's Side Architecture of P2P Cubit Connect

CUBIT identifies geometries by assigning each of them with a unique integer ID. Whenever an additional command is executed upon these geometries, the command string generated refers to that geometry's original integer ID. A problem arises since these integer IDs are sequentially assigned in the order of command execution. When one client updates the commands from other users, the CUBIT core will assign IDs that may be different from the IDs on another user's machine. Therefore, further command execution will result in significantly different models and meshes. At present, CUBIT does not have the ability to assign or reassign these numbers artificially.

With this limitation in mind, the CS prototype was developed. In the CS model, only the user-specific commands are sent to the user's core, and all peer-relevant commands are sent to a central server. The architecture of the CS prototype on the client's side can be found in Figure 4-3. The user-specific commands do not alter the model and therefore, do not affect the ID system of CUBIT. The server then distributes all the commands in the order they are received to all clients for execution (Figure 4-4). By executing commands in the same order on all clients, the models are kept consistent among multi-users. The downside to this synchronous method is that users do not have the ability to work uninterrupted without receiving other users' updates. The CS version of CUBIT Connect was used for further testing and feature implementation for the purposes of this thesis. The complete implementation details of the CS architecture can be found in Appendices B.3 and B.5.

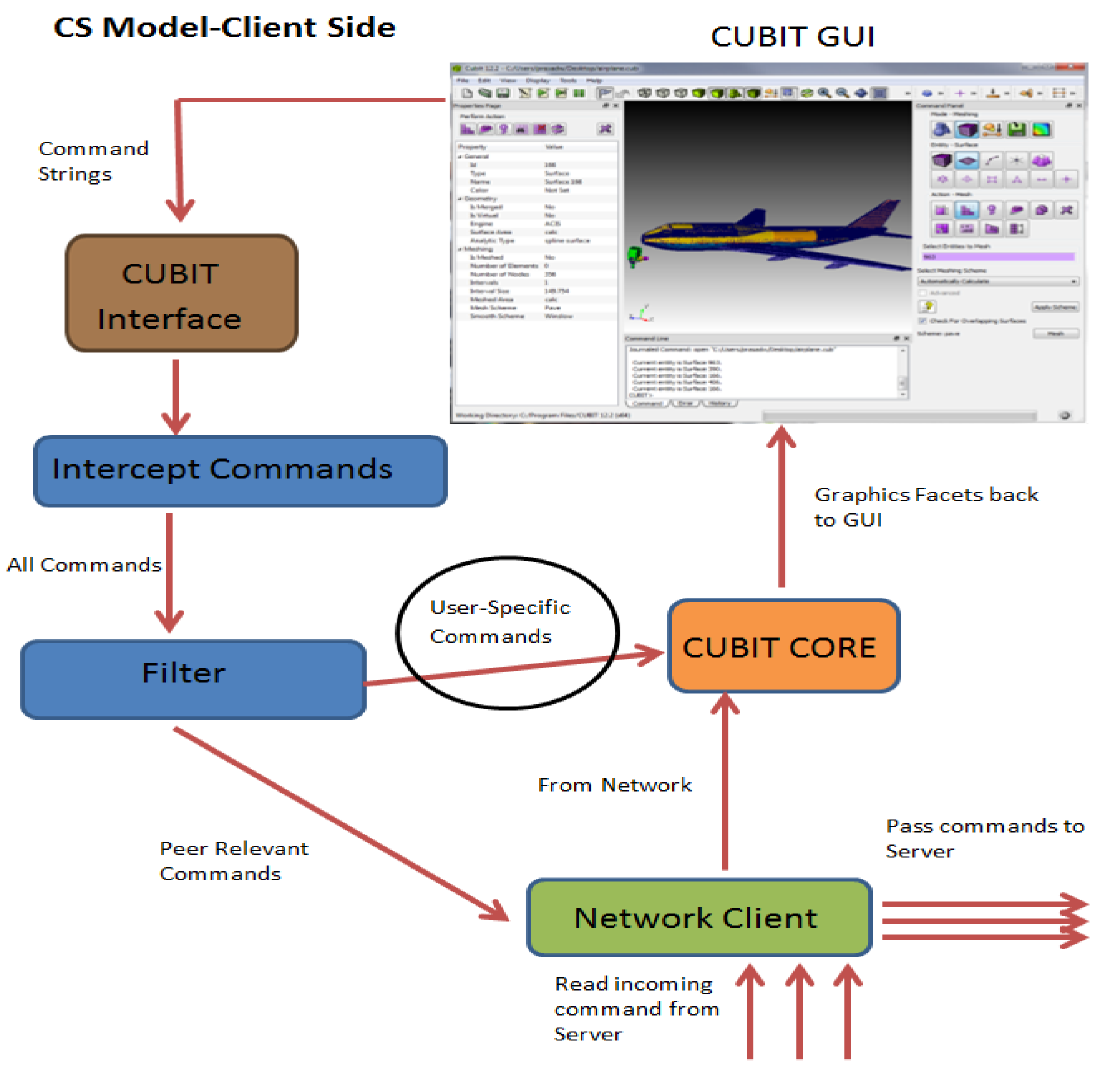


Figure 4-3 Client's Side Architecture of CS Cubit Connect

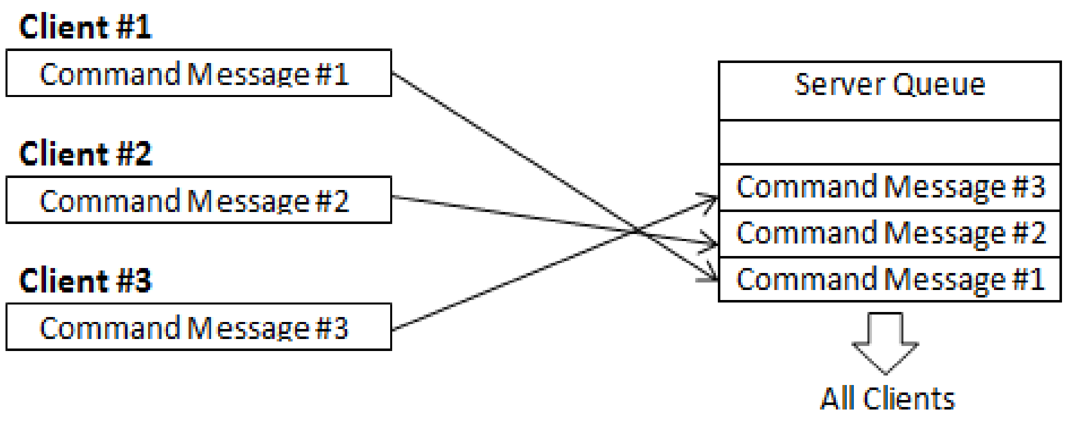


Figure 4-4: CUBIT Connect CS Message Handling Scheme

It is understood that users may not need to use the multi-user mode for everything they do with an FEA program. Therefore, the ability to switch between multi-user mode and the single-user mode can be useful. With this in mind, CUBIT Connect is programmed to allow the user to switch between multi-user mode and the single-user mode with a click of a button as shown in Figure 4-5. It is important that users do not modify models individually that are also being worked on by others in the multi-user environment. In single-user mode, the original single-user architecture is brought back while temporarily suppressing the multi-user architecture.

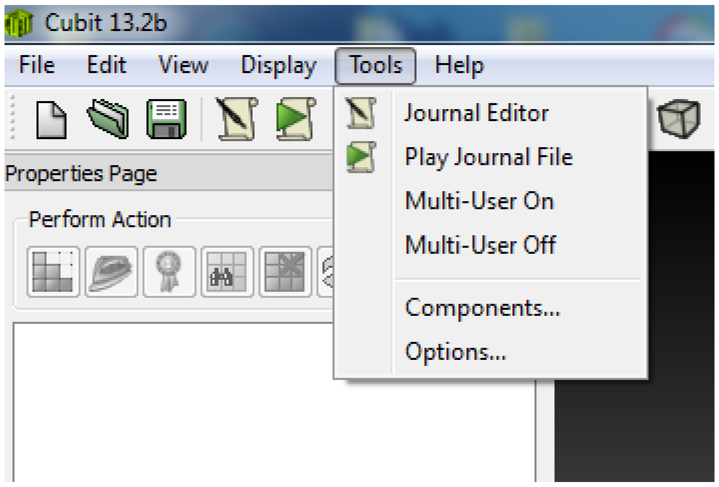


Figure 4-5: Switching between Multi-User Mode and Single-User Mode

4.3 Networking Architecture

The CS version of CUBIT Connect utilizes Windows Named Pipes (NP) (Microsoft Developer Network 2012) for inter-process communication (IPC) and TCP/IP sockets for network communications. Two networking clients (External & Internal Clients) reside on each local computer. This will be discussed in detail in section 4.3.1. C# was chosen to program the External Client (EC) that runs on the local computer as well as to program the server. C# was used as the programming language because of its superior network programming abilities. The basic architecture of CUBIT Connect is shown in Figure 4-6.

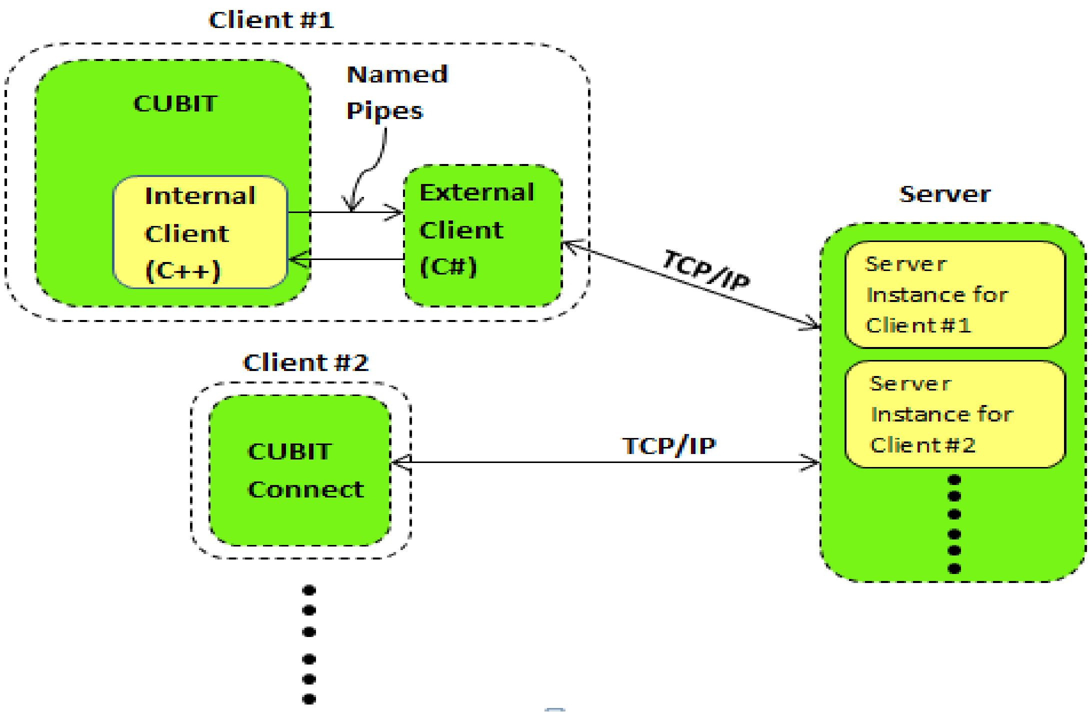


Figure 4-6: CUBIT Connect CS Architecture Utilizing both Named Pipes and TCP/IP Clients

4.3.1 Client

As mentioned above, the Client consists of two separate programs; Internal Client(IC) which is built into the source code of CUBIT and the EC which runs outside of CUBIT. To facilitate the communication between the IC and the EC, two NP's are created as illustrated in Figure 4-7.

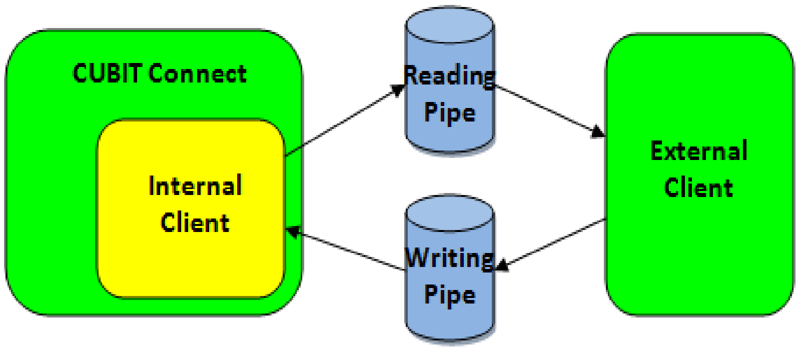


Figure 4-7 Named Pipes Connection on Each CUBIT Connect Local Machine (Client)

<u>Internal Client</u>

The IC is written in C++ and has two main functions.

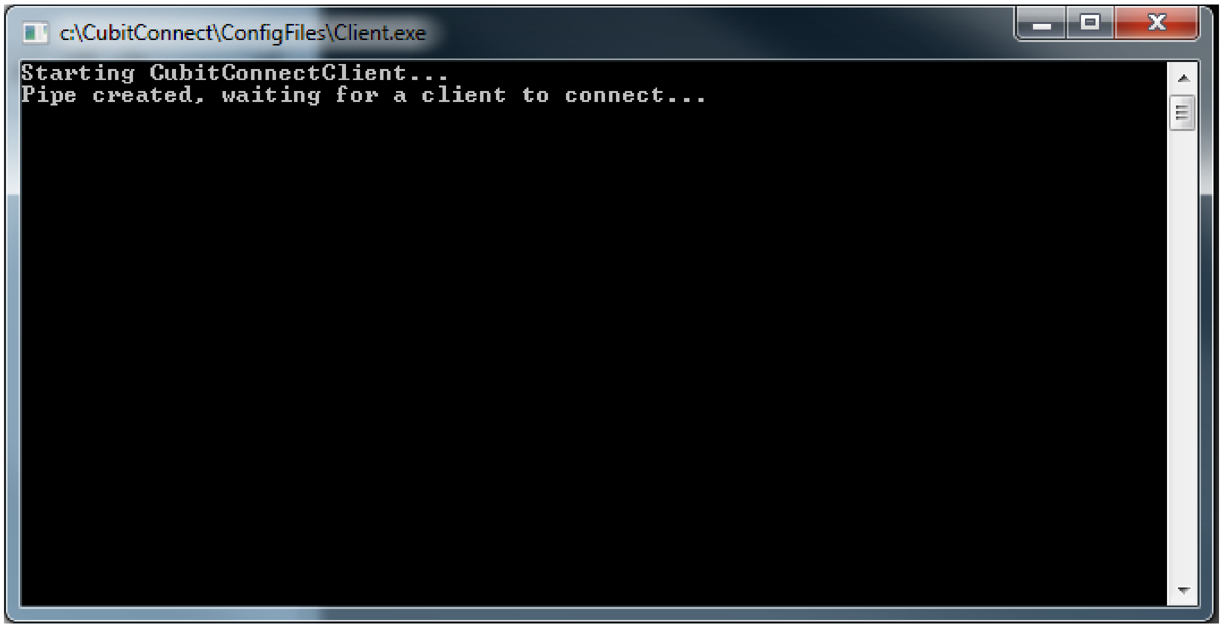
1. Intercept the command strings from CUBIT and send them via NP to the EC
2. Gather incoming multi-user commands from the EC and execute them in CUBIT

The IC is built directly into the CUBIT source code. It consists of two running threads, one dedicated to sending commands and the other dedicated to receiving. Whenever a client computer connects to both reading and writing pipes generated by the EC, it will create a Client Listening Thread (CLT) dedicated to checking the pipe for incoming messages from the EC. While the main thread is constantly sending unfiltered messages, the CLT is constantly reading for incoming messages in the writing pipe as shown in Figure 4-7. If there is a message, the reading thread will immediately place the message inside of a Client Queue (CQ) for CUBIT Connect to extract and process. This constant process is placed in a while loop, and the IC will constantly go through the CQ and update accordingly.

External Client

The EC is where the majority of client identification and message categorization takes place. Every time the EC is executed (Figure 4-8), it generates reading and writing pipes that wait for the internal client to finish the network hand shaking process. Once the pipes are established, it will initiate the connection to the central server through TCP/IP sockets. All of these processes have to be done sequentially in order to prevent any race conditions.

The EC is not only an important transition point between CUBIT and the server, but it is also where different types of messages are organized through a serialization process. Here, different message types such as the client's unique ID and the original message are combined into message structures. Some of the common message structures established in the EC are command message, master trigger, and database reset. Command messages are messages generated from the CUBIT GUI for the CUBIT core to process (e.g. "create sphere" or "mesh volume 1").



```
Starting CubitConnectClient...
Pipe created, waiting for a client to connect...
```

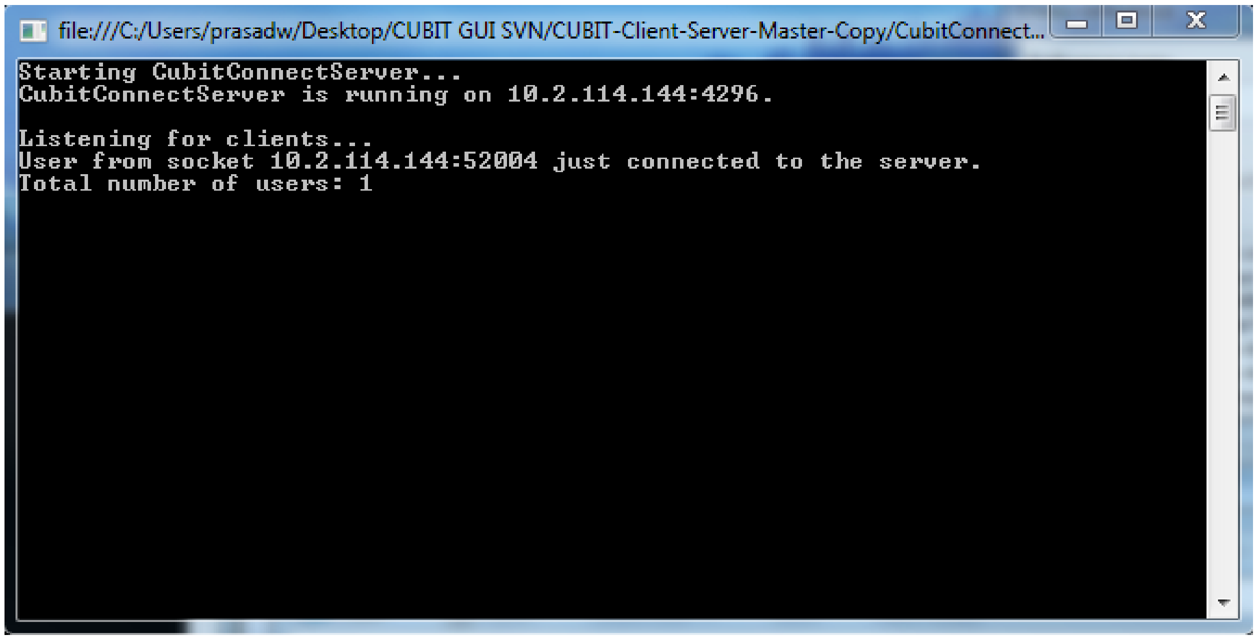
Figure 4-8: External Client Instance on the Local Machine

Master trigger messages are messages initiated by the user to re-synchronize their model with the master database (e.g. in situations such as when a client computer encounters unexpected difficulties, or when clients join the work environment at different times). Database reset messages are messages created by the user to clean up the master database (e.g. in situations where an unexpected difficulty is encountered during the collaboration process which requires the entire project to be re-established). These message structures help the server to distinguish between different CUBIT Connect operations so it can respond accordingly.

4.3.2 Server

In the global scale, WAN (Wide Area Network) communication is handled through TCP/IP sockets, where a centralized server provides sockets for client computers to initiate connections. The basic functionalities of the CUBIT server are generating TCP/IP sockets, storing messages to the database, and sending commands back into the TCP stream. Whenever a client connects to the server, a Client Thread (CT) is generated, which is separate from the main thread. While the separate CT runs on its own, the main thread loops and creates another instance of the TCP/IP stream that waits for new clients to connect. For this design, there are no limits assigned to neither thread nor pipe instances, which means the server is capable of handling as many users as the network and resources allow.

While the main thread waits for more clients, the established CT immediately listens for client messages from the TCP/IP stream. Once a message comes through the stream, it will be processed and stored into a master database. After message processing is complete, the server immediately puts the message into the stream for all the clients to pick up. The constant communication between the server and client is established through an infinite while loop and the process will not terminate unless the client terminates the connection or a network error occurs. Further networking implementation details are found in Appendices B.3 and B.5. Figure 4-9 shows an instance of the CUBIT Connect Server with one client computer connected.


file:///C:/Users/prasadw/Desktop/CUBIT GUI SVN/CUBIT-Client-Server-Master-Copy/CubitConnect...

```
Starting CubitConnectServer...
CubitConnectServer is running on 10.2.114.144:4296.

Listening for clients...
User from socket 10.2.114.144:52004 just connected to the server.
Total number of users: 1
```

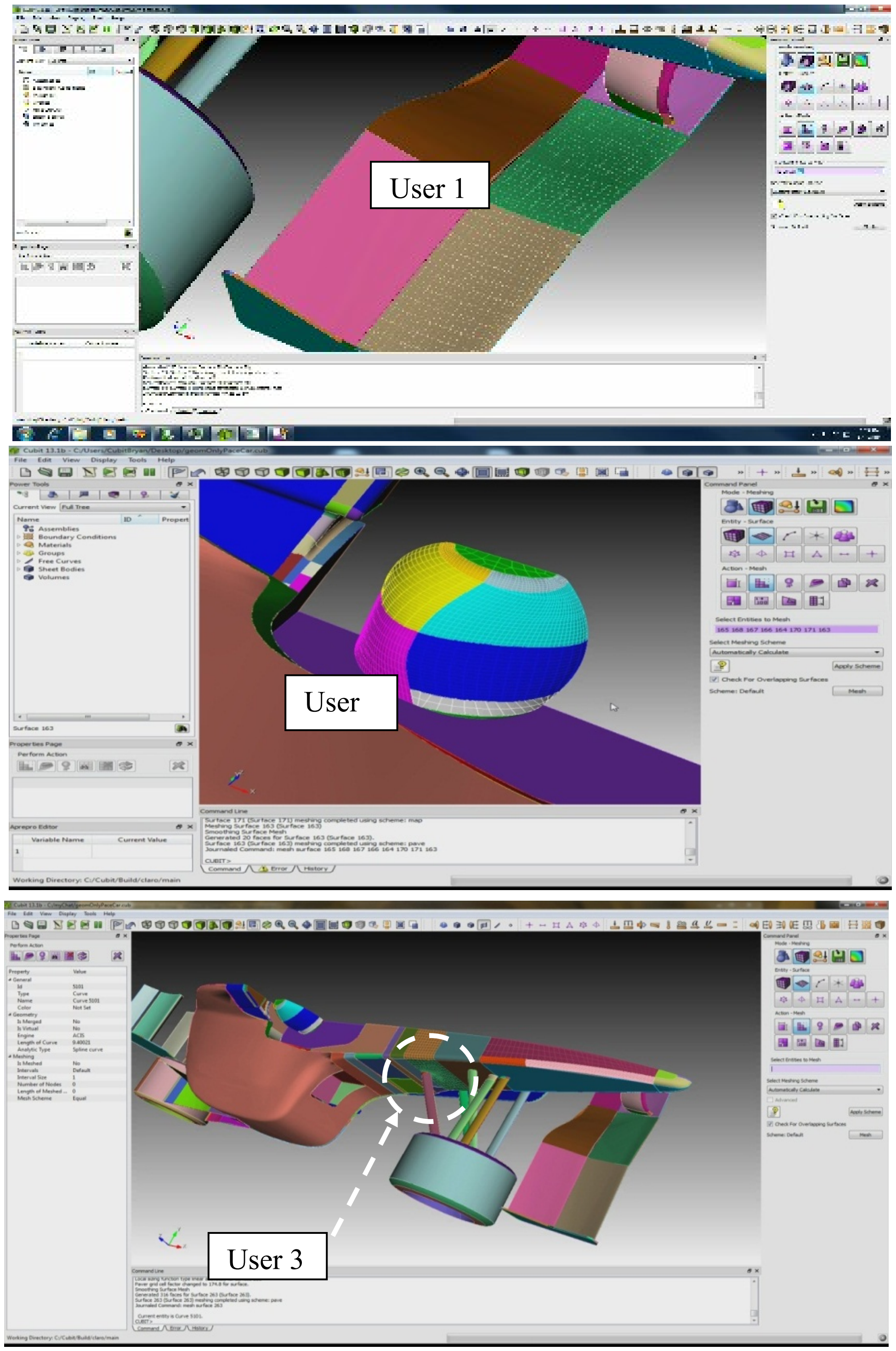
Figure 4-9: CUBIT Connect Server Instance

4.3.3 Current Capabilities of the Networking Architecture

The current capabilities of CUBIT Connect Client and Server include:

1. The ability to handle multiple projects at a given time
2. Distinguishing between different users and user levels (admin, normal user, etc.)
3. Secure login using a username and password
4. A comprehensive SQL database (command string, user credential storing, etc.)
5. Workspace assignment (section 4.4)

Figure 4-10 shows three users simultaneously editing the elements of a race car, each assigned a different region.



```
Surface 171 (Surface 171) meshing completed using scheme: map
Meshing Surface 163 (Surface 163)
Smoothing Surface Mesh
Generated 20 faces for Surface 163 (Surface 163).
Surface 163 (Surface 163) meshing completed using scheme: pave
Journaled Command: mesh surface 165 168 167 166 164 170 171 163

CUBIT>
```

```
Paver grid cell factor changed to 174.8 for surface.
Smoothing Surface Mesh
Generated 316 faces for Surface 263 (Surface 263).
Surface 263 (Surface 263) meshing completed using scheme: pave
Journaled Command: mesh surface 263

 Current entity is Curve 5101.
CUBIT>
```

Figure 4-10: CUBIT Connect Multi-User Session

4.4 **Multi-User Workspace Decomposition**

In a multi-user environment, it is necessary to assign different areas (workspaces) for different users to work on. Causally assigning a workspace to a user may not be effective, because it is likely the user will accidently cross into another user's workspace. That means, assigning workspaces without physically restricting users to certain workspace boundaries in the model would not be effective. This would ultimately create chaos and could make the collaborative environment less productive. Therefore, users should be restricted to their own workspaces and they should not be allowed to make changes to areas of the model that do not belong to their workspace. This does not mean that users should be barred from viewing the entire model. It only means the user cannot make any modifications to the areas to which they are not assigned.

It is proposed to have different user levels in a multi-user environment. The role of the user would determine which areas of the model he/she has access to. For this prototype, two user groups were created: administrator and ordinary user. An administrator has access to the entire model and is the one who assigns workspaces to other users. The ordinary user only has access to an assigned portion of the model and cannot modify other regions of the model. A GUI was setup for managing and assigning user workspaces.

4.4.1 Workspace Implementation in CUBIT Connect

As illustrated in section 3.3, a model or assembly can be decomposed using geometry tags. These tags can also be used to assign user workspaces and restrict users from modifying the workspaces of others. This method gives a user the ability to view regions of the model that belong to other users, without being able to modify any geometry or other entities in those regions.

The geometry ID scheme in CUBIT is similar to that of the scheme illustrated in Figure 3-6 and it is explained further in Table 2.

Table 2: CUBIT's Entity ID System

Geometry Tag	**Description**
Vertex	A point
Curve	A collection of points
Surface	A collection curves
Volume	A collection of surfaces
Body	A collection of volumes

Using CUBIT's numbering system, regions can be assigned as well as restricted. For this prototype, volume ID's were used to decompose a model or assembly. As illustrated in Table 2, volumes consist of vertices, curves, and surfaces. It should be noted that workspaces can be restricted using any of the aforementioned geometry ID tags. However, it would be the most effective to use surfaces and volumes in CUBIT. If needed, any combination of the geometry IDs can be used simultaneously. However, the type of geometry tag to be used in model decomposition actually depends on the specific model and its topology. As show in Figure 4-11 these geometry tags can be used to easily distinguish between regions of a racecar model. The racecar model is divided among three users where user 1 is assigned the wing areas, user 2 is assigned the wheel areas, and user 3 is assigned the nose cone area.

To allow workspace restriction, the CUBIT source file GGeomPicker.cpp (Appendix A.3), was modified. Whenever the user clicks on the model, a function checks to see what volume ID the clicked-on geometry belongs to. If that volume ID is on the list of permitted volume IDs for that user, the function lets the mouse click go through, selecting the geometry.

Otherwise, the mouse click is not allowed to go through and the user is barred from selecting the area of the model.

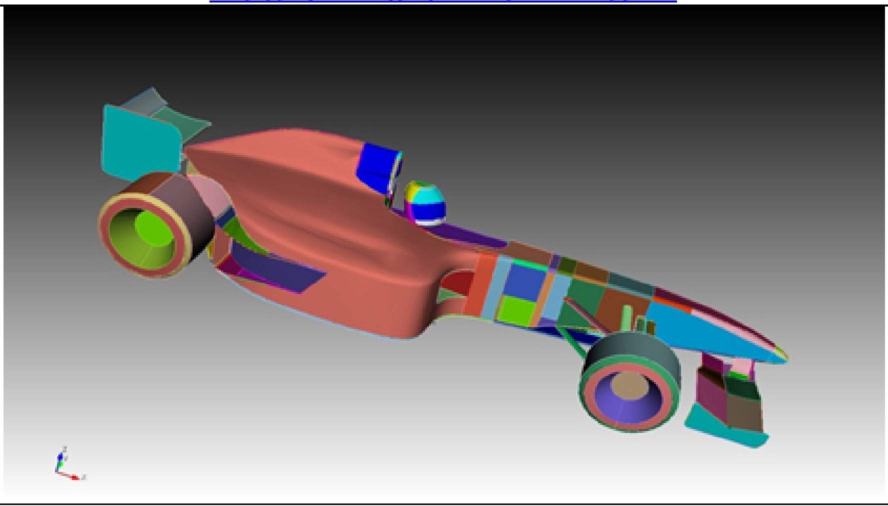
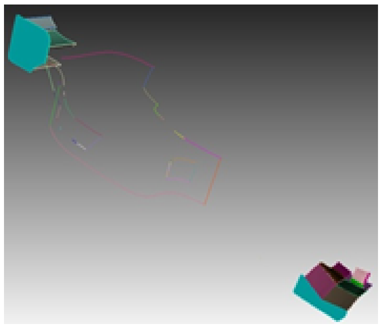
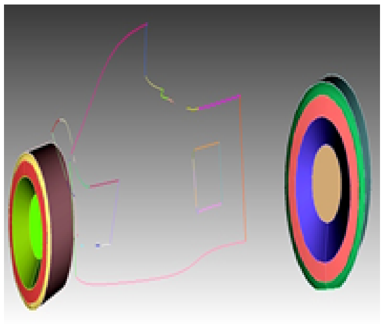
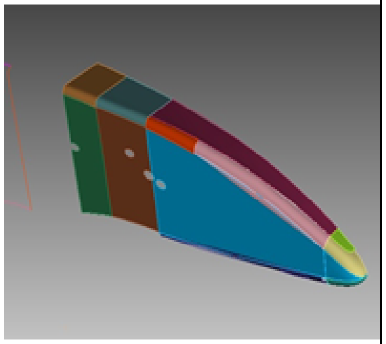
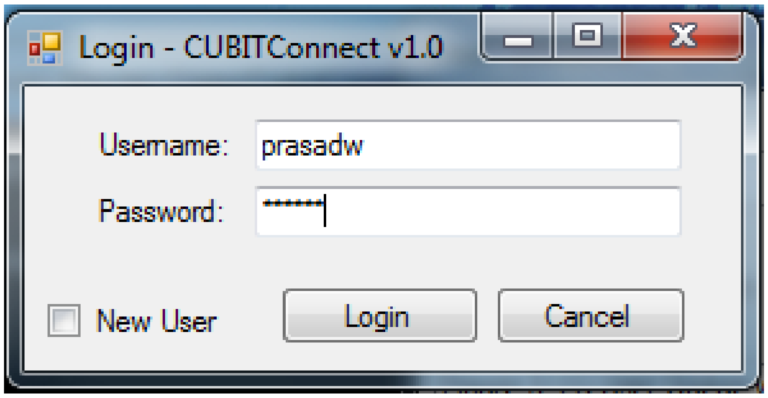
MASTER MODEL (PACE CAR) http://byuracing.byu.edu/content/pace		
		
Decomposed Work Spaces in CUBIT		
User 1	**User 2**	**User 3**
Wings Only allowed to interact with volume IDs Front Wing (IDs:44-50, 28-32, etc.) Tail Wing (IDs:354-357, 122, 379, etc.)	**Wheels** Only allowed to interact with volume IDs Front Wheel (IDs:258, 357-360, 130, etc.) Back Wheel (IDs:363-366,147, etc.)	**Nose Cone** Only allowed to interact with volume IDs : 21,22, 25,238, 239. 264, 381,etc.
		
User 1 can **select only** the volumes that are contained within the **wings** sections of the car model	User 2 can **select only** the volumes that are contained within the **wheels** sections of the car model	User 3 can **select only** the volumes that are contained within the **nose cone** section of the car model
All users can see the entire car model regardless of their assigned regions. If they wish, users can isolate the assigned regions as shows in the 3 screenshots above.		

Figure 4-11 CUBIT Connect Workspace Assignment

4.4.2 Workspace Handling GUI

A series of GUI's were added to CUBIT Connect to handle and streamline workspace assignment. Figure 4-12 shows the User Login GUI (ULGUI) that checks user credentials and allows the user to proceed to select a model or part to work on. This also checks what level of user rights (administrator, normal user, etc.) the current user is assigned from the information stored in the server database. Furthermore, new users are given the ability to add their information to the database using this GUI.

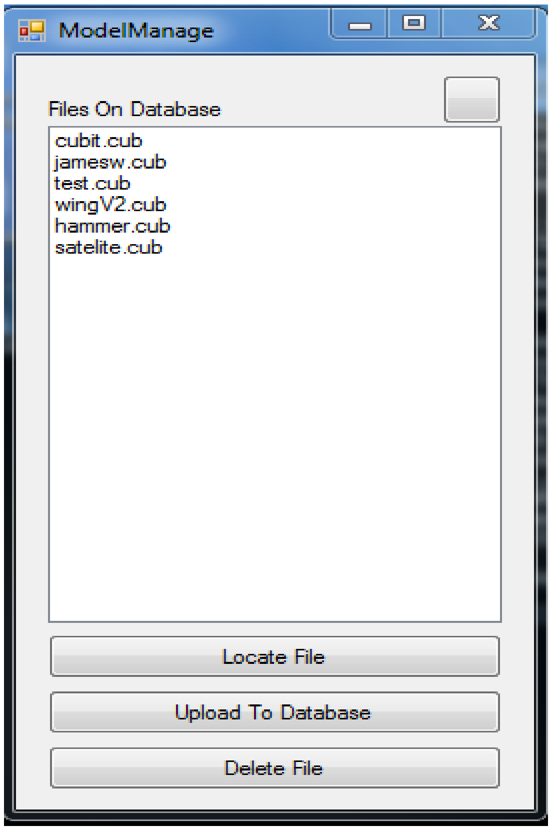


Username: prasadw
Password: ******

Figure 4-12: User Login GUI

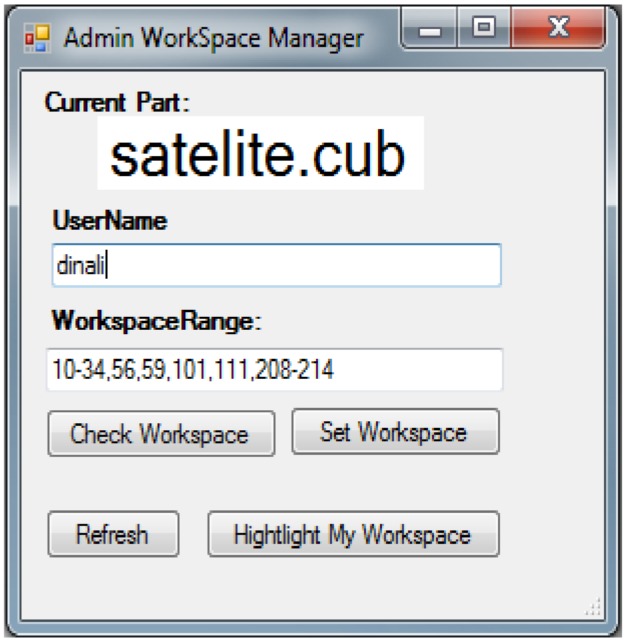
After the user successfully logs in, the Model Management GUI (MMGUI) pops up (Figure 4-13). The MMGUI then queries the server for a list of models/parts currently stored on the server and displays them. The user can then locate the file that he/she would like to work on the local computer and open it. The server then routes all the multi-user commands associated to that specific model from other multi-users working on the same model.

In the event that the user is an administrator, the Admin Workspace Manager (AWM) GUI comes up. This GUI lets the administrator assign workspaces to different users (Figure 4-14). The administrator has to input the username of the user they are assigning the workspaces to as well as the volume ID numbers to which that user has access. He/she then has the option to check the workspaces they just assigned by highlighting those in the model using the "Check Workspace" button. The "Set Workspace" button sends the user and workspace information to the server. Detailed implementation information of the workspace GUIs can be found in Appendix C.



cubit.cub
jamesw.cub
test.cub
wingV2.cub
hammer.cub
satelite.cub

Figure 4-13: Multiple Project/Model Management GUI



Current Part:
satelite.cub
UserName
dinali
WorkspaceRange:
10-34,56,59,101,111,208-214

Figure 4-14: Workspace GUI for Administrator

If the user is an ordinary user, the dialog box shown in Figure 4-15 comes up. This queries the workspace data that is already stored on the server for that particular user and saves that data to a configuration file stored on the local computer. This file is then read by GGeomPicker.cpp (Appendix A.3) to check mouse interaction authorizations. The user can use the "Highlight My Workspace" button to highlight the area of the mode they have access to as shown in Figure 4-16.

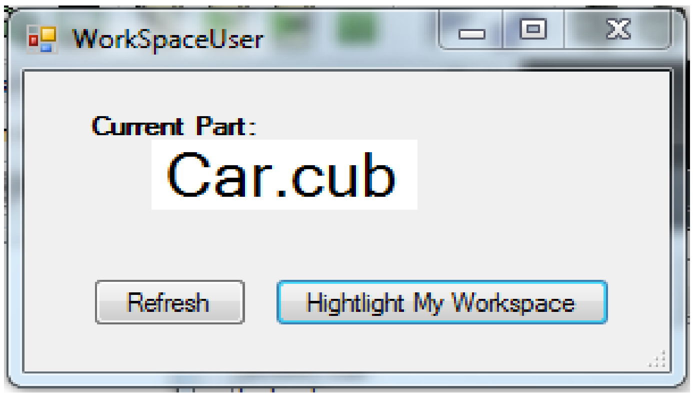


Figure 4-15: Workspace GUI for Normal User

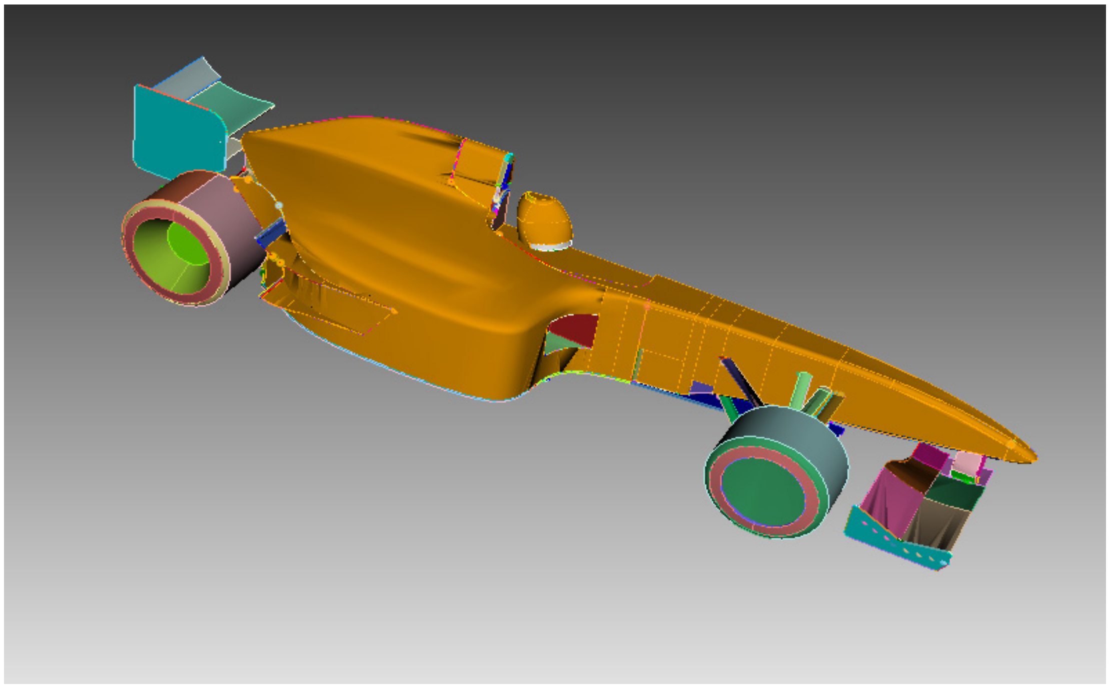
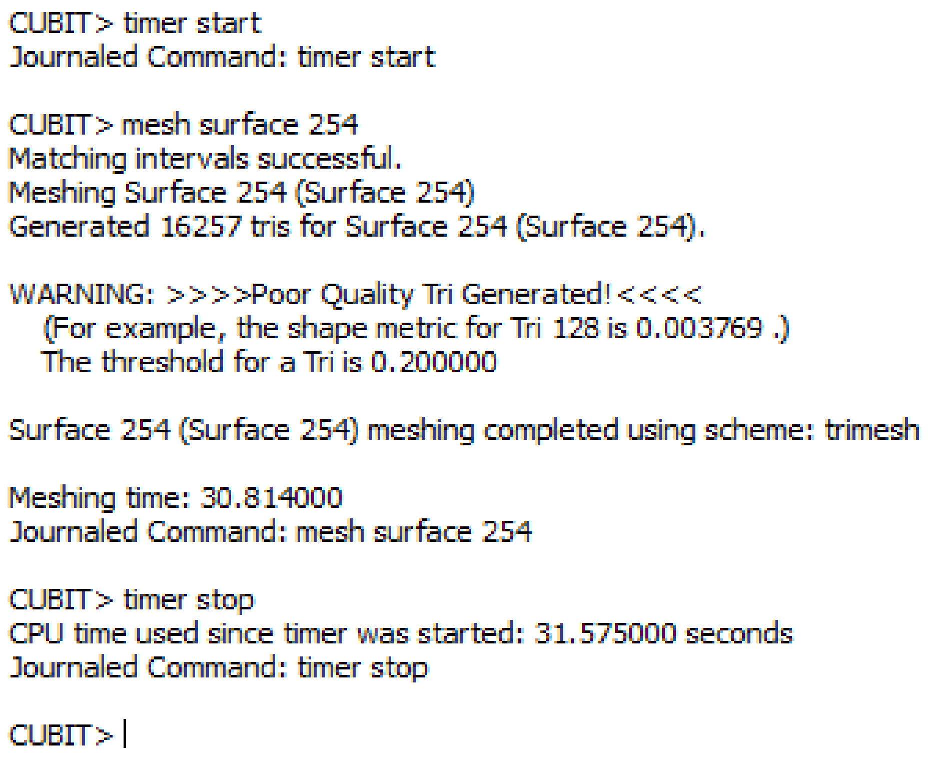


Figure 4-16: Workspace Highlight; The User has Access to the Body Area of the Car Only

4.5 Time-Consuming Task Handling

As mentioned in Section 3.4, time-consuming task handling becomes an important issue in an asynchronous FEA environment. However, since the current architecture of native CUBIT does not allow a successful implementation of an asynchronous environment, a comprehensive implementation of the time-consuming task handler was not attempted. The reason being that, in an asynchronous CUBIT Connect environment all of the users' models cannot be kept consistent.

A timer function, allows the user to see how much CPU time was utilized for a certain algorithm to run. Figure 4-17 shows an example of how the timer function works in CUBIT.



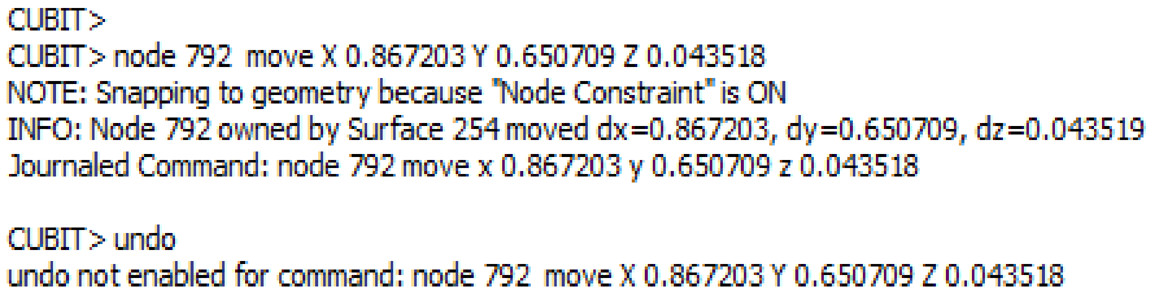
```
CUBIT> timer start
Journaled Command: timer start

CUBIT> mesh surface 254
Matching intervals successful.
Meshing Surface 254 (Surface 254)
Generated 16257 tris for Surface 254 (Surface 254).

WARNING: >>>>Poor Quality Tri Generated!<<<<
   (For example, the shape metric for Tri 128 is 0.003769 .)
   The threshold for a Tri is 0.200000

Surface 254 (Surface 254) meshing completed using scheme: trimesh

Meshing time: 30.814000
Journaled Command: mesh surface 254

CUBIT> timer stop
CPU time used since timer was started: 31.575000 seconds
Journaled Command: timer stop

CUBIT> |
```

Figure 4-17: CPU Timer Function in CUBIT

The user calls the timer function in the CUBIT command windows by typing “timer start”. The user then runs the desired meshing command and after the command is completed, the user stops the timer function by typing “timer stop” in the command window. Utilized CPU time is then displayed on the command window. This time can be captured and sent to the server with the respective command. Then a GUI that is similar to the one discussed in section 3.4 can be implemented.

4.6 Multi-User Undo Handling

As mentioned in section 3.5, CUBIT’s undo system does not allow the undoing of mesh creation, moving of nodes, or any other major meshing functions (Figure 4-18). Undo is turned off by default and if the user wants to use the undo feature, they have to turn it on. Undo works for creating and changing geometry only. However, for meshing needs, a delete function is used in the place of undo. When the delete function is called, it goes ahead and removes the specified mesh or node. The delete function works well in the multi-user mode. Therefore, using the delete function to undo changes is recommended. Since most commercial FEA programs lack an undo system for mesh related tasks, an undo handling system was not developed for this thesis.



```
CUBIT>
CUBIT> node 792  move X 0.867203 Y 0.650709 Z 0.043518
NOTE: Snapping to geometry because "Node Constraint" is ON
INFO: Node 792 owned by Surface 254 moved dx=0.867203, dy=0.650709, dz=0.043519
Journaled Command: node 792 move x 0.867203 y 0.650709 z 0.043518

CUBIT> undo
undo not enabled for command: node 792  move X 0.867203 Y 0.650709 Z 0.043518
```

Figure 4-18 CUBIT Undo Example

4.7 Time Comparison between Single-User and Multi-User CUBIT

A CAD-generated turbine engine front frame (Figure 4-19) was used to calculate the time saving advantages of single-user verses multi-user FEA pre-processing. A single user was given the CAD-generated model and asked to generate crude surface meshes covering the entire surface of the front frame with CUBIT. The single user took 19 minutes and 33 seconds to mesh the entire model with the aforementioned meshes (Figure 4-20).

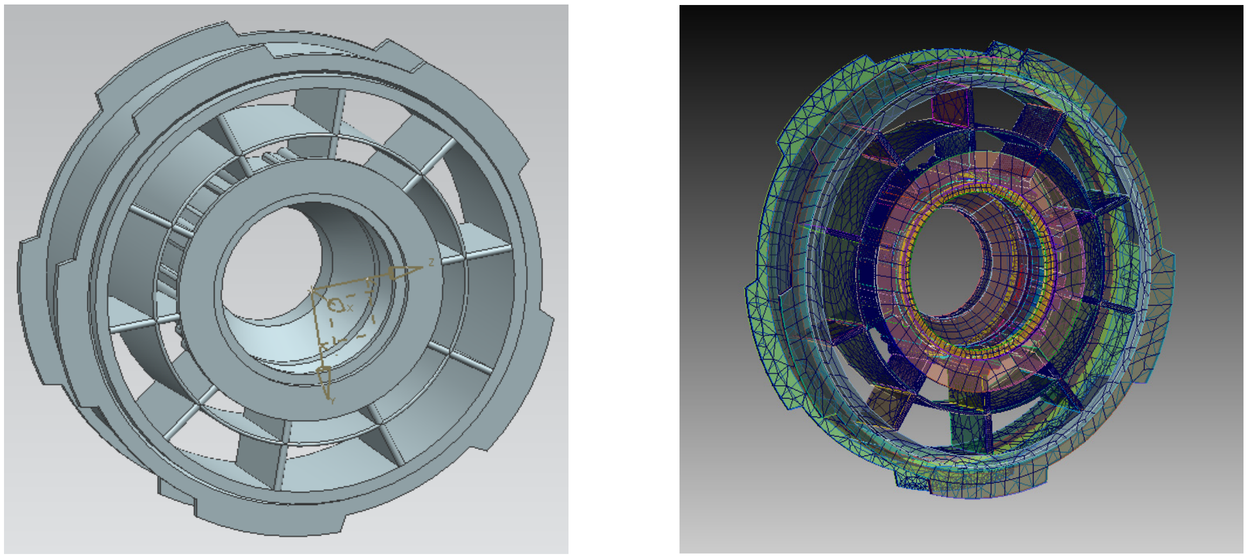


Figure 4-19: CAD-Generated and Meshed Models of Engine Front Frame

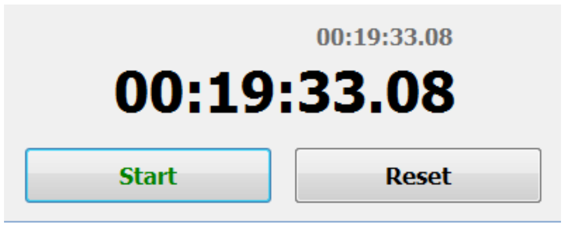


Figure 4-20: Single-User Mesh Completion Time

Three users were assigned three different areas of the front frame and were asked to generate crude surface meshes on their assigned areas using the multi-user CUBIT Connect. These three areas collectively form the entire front frame and they are shown in Figure 4-21; the times each user took to completely mesh their areas are shown in the figures as well.

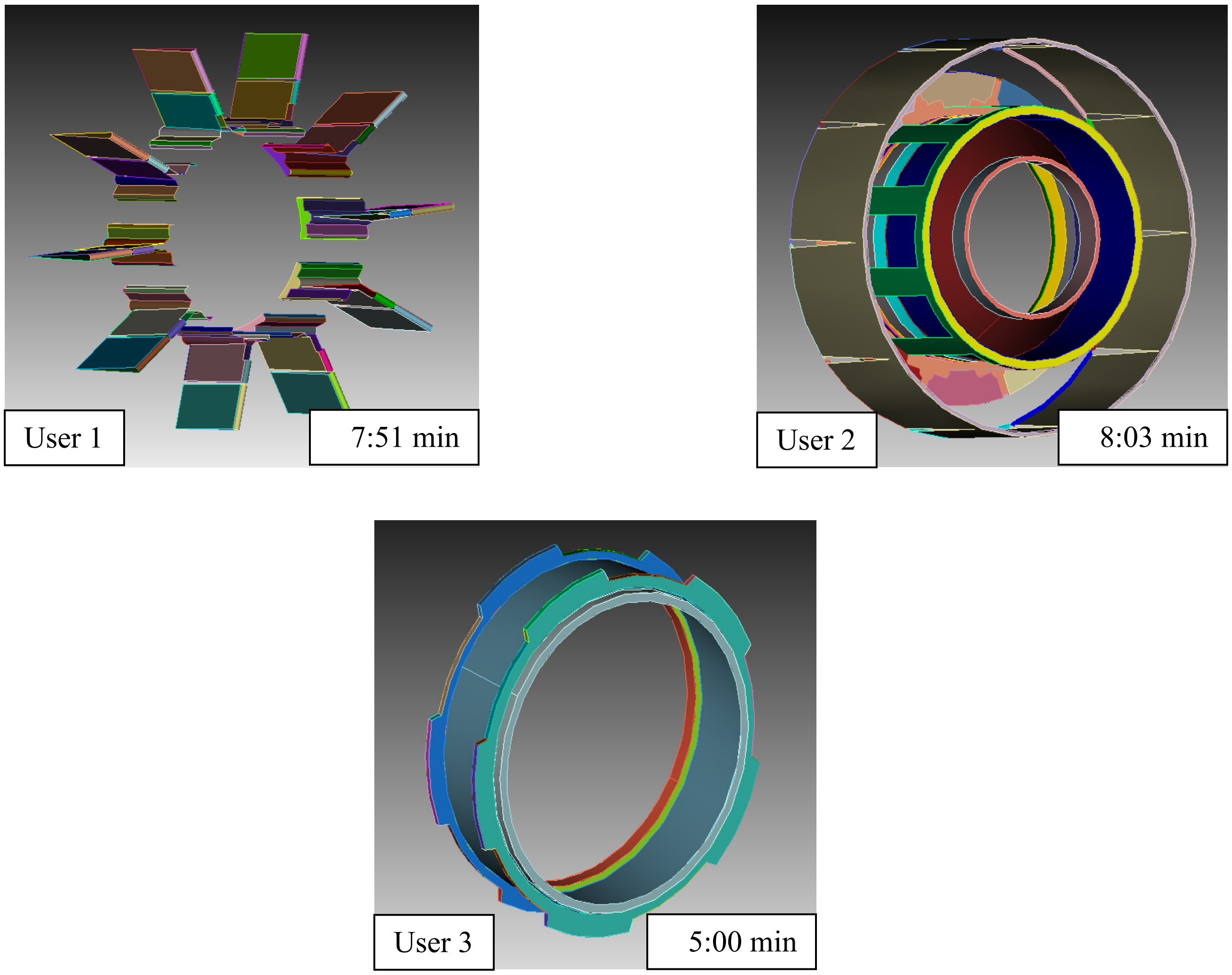


Figure 4-21: Pre-Assigned Workspaces for Front Frame 3-User Demonstration

The total time it took for the three users to complete the entire model was 08 minutes and 03 seconds (Figure 4-22), while the total number of minutes spent by all three users was 20 minutes and 54 seconds. The time to complete the model was reduced by 11 minutes and 30 seconds, which is about 60% less time than was needed with the single-user method. These results are summarized in Table 3.

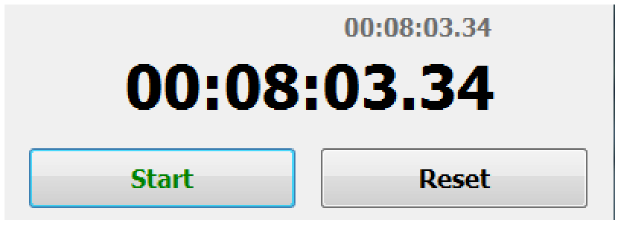


Figure 4-22: Multi-User (3 Users) Mesh Completion Time

Table 3: Meshing Time Comparison between Single-User and Multi-User FEA Pre-Processing

User Type	**Time (Minutes)**
Single User	19:33
Multi-User 1	12:44
Multi-User 2	8:03
Multi-User 3	5:00
Total Multi-User (Model completion)	8:03
MU Time Saving	11:30 (58.8%)

5 CONCLUSIONS AND RECOMMENDATIONS

Mesh generation remains the bottleneck in the modeling process of PD cycle for very large models. Often times, mesh clean up and geometry preparation may take up to 75% of the total FEA process. A robust multi-user mesh generation environment, as outlined in this thesis, has the potential to dramatically reduce this time and thus, significantly improve the efficiency of the PD process.

Furthermore, model decomposition and workspace assignment in a multi-user FEA pre-processor will change the current paradigms that depend on single-user tools. This thesis has shown that it is both possible and feasible to decompose a multi-user model using geometry IDs while restricting users to only their assigned regions. It was also shown that using three users as opposed to one user to complete a meshing task, a time saving of about 60% was accomplished.

5.1 Conclusions

This research fully meets three of the five objectives stated in the introduction and the other two objectives were met partially. This was mainly due to the limitations in the software tools used. A brief summary of the final objectives follow:

1. A generalized method for creating a multi-user FEA pre-processor was outlined in section 3.1. By using that method a robust and stable multi-user environment with full functionality was developed using CUBIT as outlined in Chapter 4.

2. A generalized method of creating a networking architecture for a multi-user FEA pre-processor was discussed in section 3.2, and complete implementations using both P2P and CS architectures were discussed in section 4.2. The CS architecture was chosen for further development and testing due to its better functionality over the P2P architecture.
3. Model decomposition using geometry ID tags was discussed in section 3.3. Using that method, a working prototype along with a comprehensive workspace management GUI was implemented as described in section 4.4
4. A method for handling time-consuming tasks in an asynchronous multi-user architecture was discussed in depth in section 3.4. Because model consistency issues arose in the asynchronous environment, this was not fully implemented. However, getting the CPU time of a completed algorithm was demonstrated in section 4.5
5. A method for handling undo in a multi-user environment was described in section 3.5. However, due to architectural limitations within CUBIT, this was not implemented.

The most challenging part of this research was working with an existing software package which has been under development for more than three decades. The program contained over 52,000 different data files and about 10 gigabytes of data. Simply put, CUBIT was not meant to be used as a multi-user program and radical changes need to be made to its architecture to make it an effective multi-user FEA pre-processor.

5.1.1 Architectural Challenges

CUBIT's GUI and core run on a single thread. For this reason, CUBIT GUI freezes while a complex and time-consuming algorithm runs in the core. This presents an enormous challenge in a multi-user environment. When users receive updates from others, they are unable to work on their model until all of those updates are executed. The GUI and core should each be made to run on separate threads in order to achieve a truly time-saving multi-user environment. Currently, CUBIT does not have the capability to utilize the multiple threads and computational cores that are available in modern Central Processing Units (CPUs). This further complicates the algorithm completion times.

The model table (geometry kernel) of CUBIT (ACIS) does not allow substantial modifications and therefore, the possibility of having a master model reside on the server is limited. ACIS is a set of closed third party libraries that do not allow modifications. Furthermore, the default ACIS operations are not thread safe. This means that all operations on the geometry kernel must be done from either a single thread or mutated in such a way that it guarantees that operations are complete on a given thread before continuing.

For the CUBIT Connect implementation, users have a local model on each computer and that model is updated to reflect changes from other users. Also, CUBIT does not have functionality to modify the entity IDs after they are created. Therefore, an asynchronous multi-user architecture, which needs to have the capability to change entity IDs, cannot be implemented.

CUBIT is written in C++ and therefore, integrating new features has to be programmed in C++ in order for them to be included in the source. Also, CUBIT takes about an hour to fully compile.

5.1.2 Workspace Decomposition

Section 4.3 describes a successful implementation of workspace restriction using entity ID’s. In this implementation, only mouse interaction is restricted to areas where the user has access. However, the user can still use CUBIT’s command line to execute changes to other areas of the model. A check has to be put in place in the “cmd( )” in CubitInteface.cpp to curb this. More research is needed to create methods to add this functionality.

The current implementation of workspace assignment only works in volume picking and surface picking modes. If the user utilizes a different mode of picking geometry (i.e. body picking, curve picking, or vertex picking), workspace assignment would fail to work properly. The method discussed in section 3.3 can be extended to these modes too. However, programming them inside of CUBIT is not straightforward and has to be studied in depth.

Some interesting questions arise from having restricted workspaces in FEA. The existence of boundaries between user workspaces makes multi-user collaboration complex. Answers to the following questions require thorough and complex research:

1. What happens at the boundary of two workspaces?
2. How can boundaries be merged in a multi-user setting so that the mesh stays consistent through the boundary?
3. How do you handle an action of a user that affects the workspace of some other user?

5.1.3 Handling of Time-Consuming Tasks

As mentioned in section 4.4, time-consuming task handling was not fully implemented in CUBIT Connect. This was because the asynchronous multi-user environment brought about complications in keeping the model consistent among users. Even after conducting in-depth research, a workaround was not found. The only solution is to have a way in CUBIT to force the entity numbering system to renumber the entities. Currently, CUBIT does not have functionality to achieve this and adding this functionality would require a significant amount of expert programming.

5.1.4 Undo Handling

As mentioned in section 4.5, the undo system of CUBIT is not fully developed and mesh related actions do not have an undo capability. Some FEA programs do not even give the user an undo option. CUBIT's undo is only functional in geometry-creation mode and it is not allowed in meshing mode. Since, the main purpose of an FEA pre-processor is to create meshes, and undo handling was not available in meshing mode, the proposed undo handling scheme was not implemented in CUBIT Connect.

5.2 **Future Work**

5.2.1 Cloud Computing FEA Pre-Processor

The cloud environment is more suitable for multi-user architectures because it eliminates having expensive hardware on the client computers, and it also streamlines global wide-area connectivity. This implementation of CUBIT should consist of a strong server and a thin client, where the CUBIT GUI resides on the client side and the core resides on the server side. That means the server will handle the computations. This would require CUBIT to be cleanly separated between the GUI and the core, so that both components can execute separately on different machines (Figure 5-1).

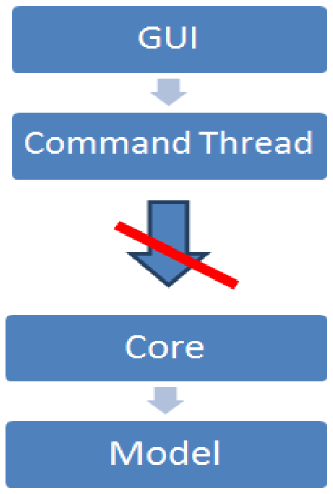


Figure 5-1: Severing Connection between CUBIT GUI & Core

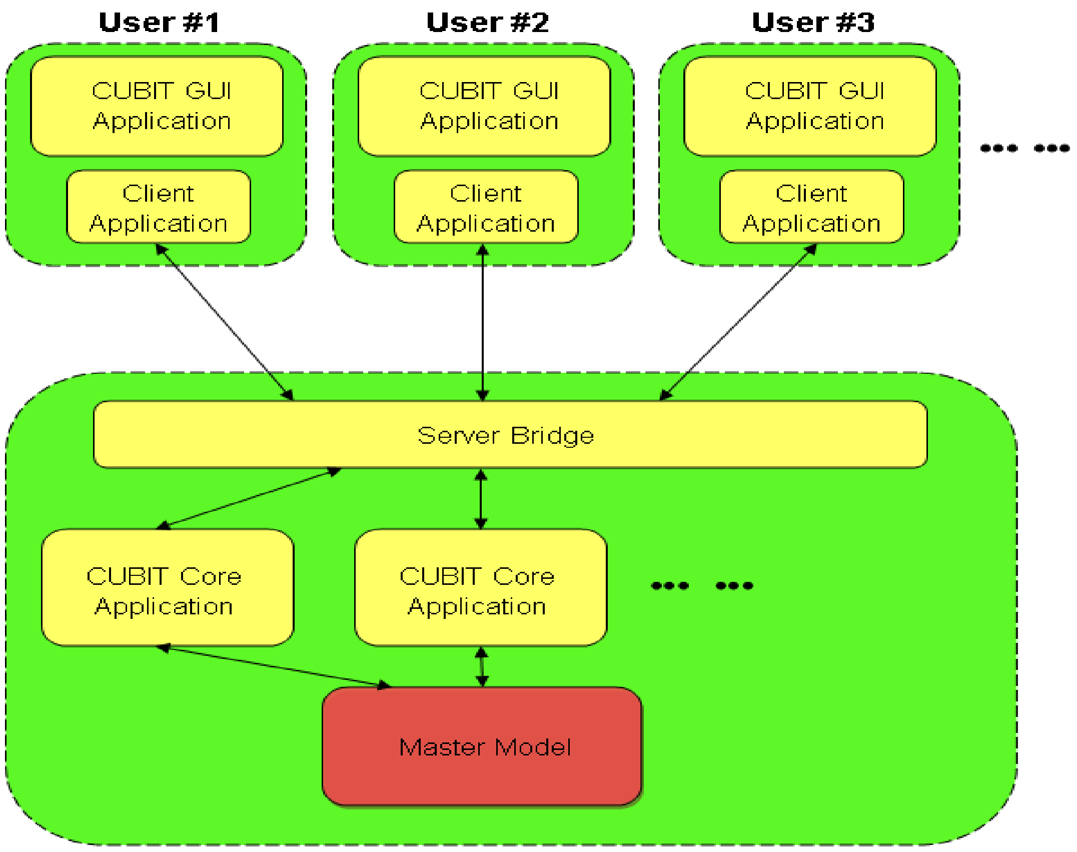


Figure 5-2: Cloud Computing CUBIT Connect Architecture

The server bridge shown in Figure 5-2 will have the capability to manage network traffic. It has to order the incoming messages into a command queue and notify an administrator when traffic is approaching the processing limit. Then it has to equally distribute the load among CUBIT cores to minimize processing times. The intention of this environment is to incorporate multiple cores capable of parallel processing. Whenever the server bridge notifies the administrator upon process limitation, a new CUBIT core can be easily created by the server to reduce the workload. This will allow the resources to be dynamically allocated, so the minimum amount of resources is used to perform the task.

At present, Sandia Labs is involved in revamping the architecture of CUBIT to separate the GUI and the core. Sandia wants a CS architecture for CUBIT with a thin client and a strong server. The final goal is to have the GUI and the graphics subsystem of CUBIT to reside on the client while the computational portion is on the server. The intent is to setup and run different types of analyses of a model/assembly concurrently on different servers. One server would run a structural analysis while another one would run a Computational Fluid Dynamics (CFD) analysis at the same time. Other types of analyses would be thermal and electromagnetic, etc. After the CUBIT architecture is modified for the multi-physics version, the implementation of the cloud multi-user environment should not take too much effort. Sandia Labs is hoping to get the separation completed by the end of year 2012.

5.2.2 HP Remote Graphics Software (RGS) Architecture

Another implementation to consider would be the HP RGS implementation where both the GUI and the Core reside on blade servers (Hewlett-Packard Development Company, L.P. n.d.). HP RGS (Figure 5-3) is a client server architecture that is especially developed to handle applications that are graphics intensive. There are three components to the HP RGS:

1. Sender – a blade workstation residing in a central facility with high end graphics capabilities
2. Receiver – a thin client with minimal processing and graphics capabilities
3. TCP/IP network – communication link between the Sender & the Receiver

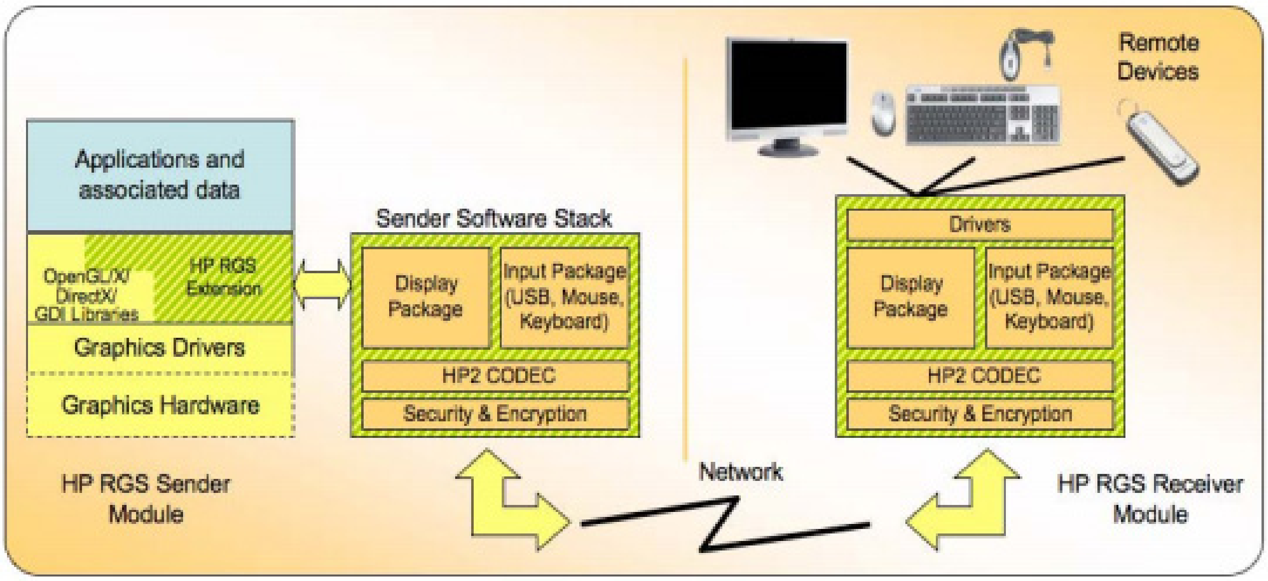


Figure 5-3: Block Diagram of HP RGS Software (www.hp.com)

The thin client establishes a one-to-one link with a blade workstation via the TCP/IP network. All the processing is done by the blade and the thin client acts only as an input terminal and a display terminal.

As a user logs on to the RGS network, the RGS Controller looks for a blade that is not being used and then assigns it to the user (Figure 5-4). As mentioned before, this creates a one-to-one link between the user and the blade. Each keystroke and mouse event is captured by the

RGS software installed on the thin client (user) and sent to the blade. Essentially, the user can control the blade from the thin client similar to that of the Microsoft's Remote Desktop Protocol (RDP) software.

For the purposes of Cubit-Connect, each blade computer has to be installed with Cubit software. There are two options for assigning a server:

1. The RGS Controller assigns a server dynamically in the blade stack
2. A blade is dedicated as a server (static)

Assigning a dedicated server would be a safe way to minimize conflicts, and dynamic server assignment should be studied further. The server then makes connections to all the blades that are using Cubit software and starts communicating. A similar approach as proposed in concept 1 can be used here to deploy Cubit-Connect.

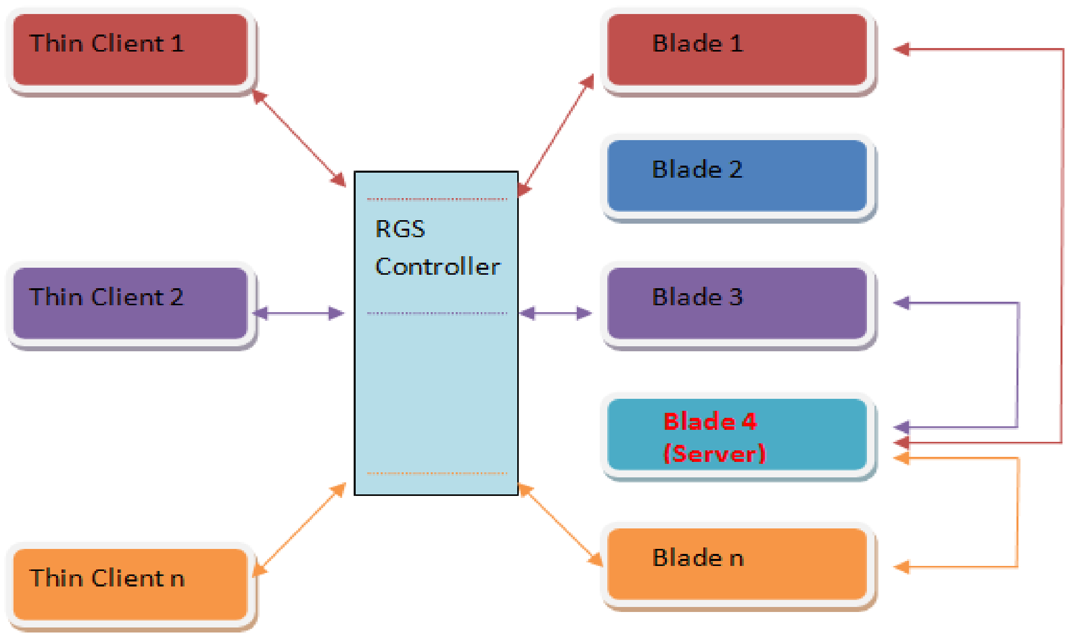


Figure 5-4: Cubit Connect in an HP RGS Setup

There are several advantages to using HP RGS software in a multi-user environment:

1. Centralized server and blades minimize security risks and protects intellectual property
2. Thin clients at the user's end do not have to be expensive computers and will not need to be upgraded regularly as their function is only to capture input and display what is being relayed by the blades.
3. Resources can be maximized and consolidated as a finite number of blades could be used and those are only assigned when there is a need.
4. High quality graphics on the inferior client computer.
5. Minimized maintenance as everything is centralized and software can be easily installed and upgraded.

Disadvantages:

1. High bandwidth usage due to graphics being tunneled through the TCP/IP connection.
2. Would have problems when used in a wide area network (WAN) as opposed to a LAN.

5.2.3 Integration with MMORPG Server

Another important step is to integrate CUBIT Connect with the MMORPG architecture Winn developed at Brigham Young University (Winn 2012). This would allow developers to add functionality to the server side without too much effort as different services could be programmed into the MMORPG server without re-writing the core server code. Also, this architecture has better data handling capabilities and would allow faster and reliable connections. With the current setup of CUBIT Connect, it should not be hard to change over to the MMORPG server.

REFERENCES

Anbo, W, G Yingsan, Z Guogang, and W Jianhua. "An agent-based collaborative FEA system for the optimal design of electromagnets." *International Conference on Power System Technology.* IEEE Xplore, 2002. 2150 - 2154.

Balling, R J. *Finite Elements.* Vol. II. BYU Academic Publishing, n.d.

Bonneau, P, and L Gabrielaitis. "Applying multi-user technology for modeling complex CAD objects." *Engineering Structures and Technologies*, 2009.

Bu, J, B Jiang, and C Chen. "Maintaining semantic consistency in real-time collaborative graphics editing systems." *IJCSNS International Journal of Computer Science and Network Security* 6, no. 4 (April 2006): 57-61.

"Cubit 13.1 User Documentation." *CUBIT.* n.d. http://cubit.sandia.gov/help-version13.1/Cubit_13.1_User_Documentation.pdf.

Douglas, S, E Tanin, A Harwood, and S Karunasekera. "Enabling Massively Multi-Player Online Gaming Applications on a P2P Architecture." *In Proceedings of the IEEE International Conference on Information and Automation.* IEEE, 2005. 7--12.

Google . *Google Docs.* n.d. http://docs.google.com.

Haghichi, K, and C Nyquist. "Introduction to Ansys." *Purdue University.* 2007. https://engineering.purdue.edu/~abe601/ansys/ansys_overview_v81_601.pdf.

Hewlett-Packard Development Company, L.P. *HP Remote Graphics Software.* n.d. http://www.hp.com/united-states/campaigns/workstations/remote-graphics-software.html#.T6QdM-tDwjU.

Lee, H, J Kim, and A Banerjee. "Collaborative intelligent CAD framework incorporating design history tracking algorithm." *Computer-Aided Design* (Elsevier) 42, no. 12 (December 2010): 1125-1142.

Marshall, F. *Model Decomposition and Constraints to Parametrically Partition Design Space in a Collaborative CAx Environment. (Master's Thesis).* Brigham Young University, 2011.

Merkley, K G. *The Tool Set for Building Claro-A Component Loading Architecture.* Sandia National Laboratories, 2006.

Microsoft Developer Network. *Named Pipes.* 2 13, 2012. http://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx.

Nidamarthi, S, R H Allen, and R D Sriram. "Observations from supplementing the traditional design process via Internet-based collaboration tools." *International Journal of Computer Integrated Manufacturing* 14, no. 1 (2001): 95-107.

Nokia Corporation. *Qt- Cross-platform application and UI framework.* 2012. http://qt.nokia.com/.

Owen, S J, et al. "An Immersive Topology Environment for Meshing." *16th International Meshing Roundtable.* Seattle, WA, 2008.

Peng, J, F Mckenna, G L Fenves, and K H Law. "An Open Collaborative Model for Development of Finite Element Program." *Proceedings of the Eighth International Conference on Computing in Civil and Building Engineering (ICCCBE-VIII}.* 2000. 1309--1316.

Red, E, G Jensen, D French , and P Weerakoon. "Multi-user architectures for computer-aided engineering collaboration." *17th International Conference on Concurrent Enterprising (ICE).* Aachen, Germany: IEEE Xplore, 2011. 1-10.

Red, E, V Holyoak, G Jensen, F Marshall , J Ryskamp, and Y Xu. "n-CAx: A Research Agenda for Collaborative Computer-Aided Applications." *Computer-Aided Design and Applications* 7, no. 3 (2010): 387-404.

Sandia National Laboratories. *CUBIT.* n.d. http://cubit.sandia.gov/.

Sriram, D, R Logcher, and S Fukuda. "Computer-Aided Cooperative Product Development." *MIT-JSME Workshop, Lecture Notes in Computer Science.* Cambridge, USA: Springer-Verlag, 1991.

SWIG. *Simplified Wrapper and Interface Generator.* 02 19, 2012. http://www.swig.org/.

v-CAx. *New Multi-User Computer-Aided Applications.* 2011. http://v-cax.byu.edu/.

Wikipedia. *Massively multiplayer online role-playing game.* March 07, 2012. http://en.wikipedia.org/wiki/Mmorpg.

Winn, J. "Integration of Massive Multi-User Online Role Playing Game Architecture with Multi-User CAx Applications (Master's Thesis)." Brigham Young University, 2012.

Xu, Y. *A Flexible Context Architecture for a Multi-User GUI (Master's Thesis).* Brigham Young University, 2010.

Xu, Y, E Red, and C G Jensen. "A Flexible Context Architecture for a Multi-User GUI." *Computer-Aided Design and Applications* 8, no. 4 (2011): 479-497.

Yu, J, J Cha, Y Lu, W Xu, and M Sobolewski. "A CAE-integrated distributed collaborative design system for finite element analysis of complex product based on SOOA." *Advances in Engineering Software* (Elsevier Science Ltd) 41, no. 4 (2010): 590-603.

Zienkiewicz, O C, R L Taylor, and J Z Zhu. *The Finite Element Method: Its Basis and Fundamentals, 6th Edition.* 6. Butterworth-Heinemann, 2005.

APPENDIX A. PROGRAMMING CODE

All the CUBIT Connect files can be found in the v-CAx SVN svn://it.et.byu.edu:1667/svn in subfolders:

CubitConnect

CUBITCONNECT TCPIP and NP

GUIDrivenCUBIT

A.1 Unaltered cmd( ) Function in CubitInterface.cpp

```
// Sends a command directy to cubit
void CubitInterface::cmd(const char *input_string)
{
  was_undoable = false;
  int ss = CubitUndoManager::number_undo_groups();
  CIUserInterface *ui = CIUserInterface::instance();
  if(ui && ui->get_playback_handler())
    ui->parse_input_line(input_string);
  else
    UserInterface::instance()->parse_input_line_and_files(input_string);
  was_undoable = CubitUndo::get_undo_enabled() &&
(CubitUndoManager::number_undo_groups() > ss);
  if(!gsObserver) return;
  std::list<CubitInterface::CIObserve*>::iterator iter = gsObserver-
>mGUIObserver.begin();
  for (; iter != gsObserver->mGUIObserver.end(); iter++)
  {
    (*iter)->notify_command_complete();
  }
}
```

A.2 Modified cmd( ) Function for Multi-User Handling

```
//Multi-User Namedpipe Implementation Initiation
PipeClient* chat = PipeClient::Instance();
ServerInterface* serverlet = ServerInterface::Instance();



// Sends a command directy to cubit
void CubitInterface::cmd(const char *input_string)
{
  was_undoable = false;
  int ss = CubitUndoManager::number_undo_groups();

  //Multi-User
 if(getMultiUser()== true)
 {

     //Check to see if the command is user-specific
      string no_cmd[] =
{"graphics","draw","list","display","quality","preview","visibility", "null",
"color", "updatecubit", "undo", "reset","open","import", "save", "highlight",
"hardcopy"};      //list of user specific commands
      int NO_CMD_LEN = sizeof(no_cmd)/sizeof(string);  //# of user-specific
commands so far

      //See if the input_string contains a user-specific command
      bool toSend = true;
      string in_str(input_string);
      for(int i=0;i<NO_CMD_LEN;i++)
      {
            if(in_str.find(no_cmd[i]) != string::npos)
            {
              //If incoming command does contain any user-specific commands,
it will do straight to the CUBIT core for execution
                  was_undoable = false;
                  int ss = CubitUndoManager::number_undo_groups();
                  CIUserInterface *ui = CIUserInterface::instance();
                  if(ui && ui->get_playback_handler())
                  {
                        ui->parse_input_line(input_string);
                  }
                  else
                  {
                        UserInterface::instance()-
>parse_input_line_and_files(input_string);

                  }
                  was_undoable = CubitUndo::get_undo_enabled() &&
(CubitUndoManager::number_undo_groups() > ss);
                  if(!gsObserver) return;

                  std::list<CubitInterface::CIObserve*>::iterator iter =
gsObserver->mGUIObserver.begin();
                  for (; iter != gsObserver->mGUIObserver.end(); iter++)
                  {
                        (*iter)->notify_command_complete();
                    }
                    toSend = false;
                }
        }
        //If It is not a user specific command, send to server
        try
        {
                if (toSend)
                {
                        PRINT_INFO("Multi User Command Sent  \n ");

                        //CUBIT sending multi-user cammands to the server
                        chat->SendMessage(input_string);
                }
                if (!toSend) PRINT_INFO("User-Specific Command Sent \n");
                //if it is not user specific, send to other people on "chat"

        }
        catch (std::exception& e)
        {
                const char* name1;
                name1= e.what();
                PRINT_INFO (name1);
                PRINT_INFO ("Exception \n");
        }

        //to receive incomming command strings
        if(in_str.find("updatecubit") != string::npos)
        {
                PRINT_INFO("Updating commands \n");
                //vector for update command
                vector <string> update_input_string_in ;
                //check client queue for upadted commands
                while(serverlet->returnQueueObject().size() > 0)
                {
                        update_input_string_in.push_back(serverlet->GetCmdQueue());
                }
                //If inputfile in not empty, send the command string to Cubit
Core for execution
                if(!update_input_string_in.empty())
                {
                        was_undoable = false;
                        int ss = CubitUndoManager::number_undo_groups();
                        CIUserInterface *ui = CIUserInterface::instance();
                        for (int i=0; i<update_input_string_in.size(); i++)
                        {
                                if(ui && ui->get_playback_handler())
                                {
                                        ui-
>parse_input_line(update_input_string_in[i].c_str());
                                }
                                else
                                {
                                        UserInterface::instance()-
>parse_input_line_and_files(update_input_string_in[i].c_str());
```

```
                        }
                        was_undoable = CubitUndo::get_undo_enabled() &&
(CubitUndoManager::number_undo_groups() > ss);
                        if(!gsObserver) return;
                        std::list<CubitInterface::CIObserve*>::iterator iter
= gsObserver->mGUIObserver.begin();
                        for (; iter != gsObserver->mGUIObserver.end();
iter++)
                        {
                              (*iter)->notify_command_complete();
                        }
                  }
            }else{
                  PRINT_INFO("Nothing in the client queue \n");
            }
      }else{

            Sleep(1000);
            //vector for regular execution
            vector <string> input_string_in;
            while(serverlet->returnQueueObject().size() > 0)
            {
                  input_string_in.push_back(serverlet->GetCmdQueue());
            }

            if(!input_string_in.empty())
            {
                  was_undoable = false;
                  int ss = CubitUndoManager::number_undo_groups();
                  CIUserInterface *ui = CIUserInterface::instance();
                  for (int i=0; i<input_string_in.size(); i++)
                  {
                        if(ui && ui->get_playback_handler())
                        {
                              ui-
>parse_input_line(input_string_in[i].c_str());
                        }
                        else
                        {
                              UserInterface::instance()-
>parse_input_line_and_files(input_string_in[i].c_str());

                        }
                        was_undoable = CubitUndo::get_undo_enabled() &&
(CubitUndoManager::number_undo_groups() > ss);
                        if(!gsObserver) return;

                        std::list<CubitInterface::CIObserve*>::iterator iter
= gsObserver->mGUIObserver.begin();
                        for (; iter != gsObserver->mGUIObserver.end();
iter++)
                        {
                              (*iter)->notify_command_complete();
                        }
                  }
            }else{
                  PRINT_INFO("Command not yet executed, please click update
\n");
            }
      }
}
else
{
  CIUserInterface *ui = CIUserInterface::instance();
  if(ui && ui->get_playback_handler())
    ui->parse_input_line(input_string);
  else
    UserInterface::instance()->parse_input_line_and_files(input_string);
  was_undoable = CubitUndo::get_undo_enabled() &&
(CubitUndoManager::number_undo_groups() > ss);
  if(!gsObserver) return;

  std::list<CubitInterface::CIObserve*>::iterator iter = gsObserver-
>mGUIObserver.begin();
  for (; iter != gsObserver->mGUIObserver.end(); iter++)
  {
    (*iter)->notify_command_complete();
  }
}

  //This section is where it gets the total number of volumes and write that
to a file
  string check_open="open";
  string in_str_check(input_string);

  if (in_str_check.find(check_open) != string::npos)
  {
       std::ofstream VolumeFile("C:\\CubitConnect\\ConfigFiles\\Volume.txt");
     VolumeFile.clear();
       int number_of_volumes= get_volume_count();
       VolumeFile<< number_of_volumes << endl;
       VolumeFile.close();
  }
}
```

A.3 CUBIT GGeomPicker.cpp Workspace Handling Implementation

```
#ifdef WIN32
#pragma warning(disable : 4786)
#endif

#include "GGeomPicker.hpp"
#include "GCGMInterface.hpp"

#include <map>

#include "vtkPoints.h"
#include "vtkRenderer.h"
#include "vtkCell.h"
#include "vtkCellData.h"
#include "vtkVariantArray.h"
#include "vtkFieldData.h"
#include "vtkDataSet.h"
#include "vtkPainterPolyDataMapper.h"
#include "CubitString.hpp"
#include "GTopEntity.hpp"

#include "MultiCellPicker.hpp"

// CGM includes

#include "GEntity.hpp"
#include "GVertex.hpp"
#include "GEntityGroup.hpp"
#include "GSurface.hpp"
#include "GCurve.hpp"
#include "GSurfaceData.hpp"
#include "SVUtil.hpp"
#include "GRepresentationPainter.hpp"
#include "GGeomModel.hpp"

//Multi-User Includes
#include <fstream>
#include <vector>
#include <algorithm>

bool MultiUserWorkSpaces(int PickedVolumeID);

static const float gVertexOffset = 0.0;
static const float gCurveOffset = 0.1;
static const float gSurfaceOffset = 0.2;
static const float gVolumeOffset = 0.3;
static const float gBodyOffset = 0.4;


const char* GGeomPicker::gGroupType = "Group";
const char* GGeomPicker::gBodyType = "Body";
const char* GGeomPicker::gVolumeType = "Volume";
const char* GGeomPicker::gSurfaceType = "Surface";
const char* GGeomPicker::gCurveType = "Curve";
const char* GGeomPicker::gVertexType = "Vertex";
const char* PickedGeomEntity::type() const
{
      return mIface->class_name(mRefEntity);
}

int PickedGeomEntity::id() const
{
  return mIface->id(mRefEntity);
}

const char* PickedGeomEntity::name() const
{
  static CubitString thename;
  thename = mIface->entity_name(mRefEntity);
  return thename.c_str();
}

size_t PickedGeomEntity::handle() const
{
  return reinterpret_cast<size_t>(mRefEntity);
}

GGeomPicker::GGeomPicker()
{
  register_picker(gGroupType, this);
  register_picker(gBodyType, this);
  register_picker(gVolumeType, this);
  register_picker(gSurfaceType, this);
  register_picker(gCurveType, this);
  register_picker(gVertexType, this);
}

GGeomPicker::~GGeomPicker()
{
  unregister_picker(gGroupType, this);
  unregister_picker(gBodyType, this);
  unregister_picker(gVolumeType, this);
  unregister_picker(gSurfaceType, this);
  unregister_picker(gCurveType, this);
  unregister_picker(gVertexType, this);
}

void GGeomPicker::ready_to_pick(vtkRenderer* ren, MultiCellPicker*)
{
}

void GGeomPicker::done_picking(vtkRenderer* ren, MultiCellPicker*)
{
}

void GGeomPicker::compute_pre_select(vtkRenderer* ren, float x, float y)
{
  MultiCellPicker* VTKPicker = MultiCellPicker::New();

  // get geometry ready to pick
  ready_to_pick(ren, VTKPicker);
  // now do picking
  VTKPicker->Pick(x,y, 0.0, ren);

  // cleanup what we got ready to pick
  done_picking(ren, VTKPicker);

  // find the geometry from our pick list
  process_picks(ren, VTKPicker, false);

  VTKPicker->Delete();
}

void GGeomPicker::compute_pick(vtkRenderer* ren, float x, float y)
{
  MultiCellPicker* VTKPicker = MultiCellPicker::New();

  // get geometry ready to pick
  ready_to_pick(ren, VTKPicker);

  // now do picking
  VTKPicker->Pick(x,y, 0.0, ren);

  // cleanup what we got ready to pick
  done_picking(ren, VTKPicker);

  // find the geometry from our pick list
  process_picks(ren, VTKPicker, false);

  // get groups out of our selection buffer if we are picking groups
  if(is_filtered(gGroupType))
  {
    process_groups();
  }

  VTKPicker->Delete();

}

void GGeomPicker::compute_pick_by_bounds(vtkRenderer* ren, PickBounds*
bounds)
{
  // TODO:  when processing picks, make sure entire datasets of curves or
  // surfaces were selected

  MultiCellPicker* VTKPicker = MultiCellPicker::New();

  // get geometry ready to pick
  ready_to_pick(ren, VTKPicker);

  // now do picking
  VTKPicker->PickByBounds(bounds, ren);

  // cleanup what we got ready to pick
  done_picking(ren, VTKPicker);

  // find the geometry from our pick list
  process_picks(ren, VTKPicker, true);

  // get groups out of our selection buffer if we are picking groups
  if(is_filtered(gGroupType))
  {
    process_groups();
  }

  VTKPicker->Delete();
}

// have a map to process surfaces at the end,
// this is in case we are selecting by box and we only want fully enclosed
volumes and bodies
// not just fully enclosed surfaces
struct DeferredSurfaceData
{
  DeferredSurfaceData(PickedGeomEntity* pick, const float* pt, float z, bool
tol)
   : mPickedGeom(pick), mPoint(pt), mZ(z), mTol(tol)
  {}
  PickedGeomEntity* mPickedGeom;
  const float* mPoint;
  float mZ;
  bool mTol;
};


void GGeomPicker::process_picks(vtkRenderer* ren, MultiCellPicker*
vtk_picker, bool box_pick)
{
  const MultiCellPicker::CellPickList* picks =
    vtk_picker->get_picked_cells();

  MultiCellPicker::CellPickList::const_iterator iter;
  MultiCellPicker::SubCellPickList::const_iterator jter;

  bool filter_types[6] = { false,false,false,false,false,false };
  if(is_filtered(gGroupType))
    filter_types[0] = true;
  if(is_filtered(gBodyType))
    filter_types[1] = true;
  if(is_filtered(gVolumeType))
    filter_types[2] = true;
  if(is_filtered(gSurfaceType))
    filter_types[3] = true;
  if(is_filtered(gCurveType))
    filter_types[4] = true;
  if(is_filtered(gVertexType))
    filter_types[5] = true;

  // TODO -- speed this up esp. for rubber band picking
  // We'd save lots of time if we didn't call add_to_selection_buffer so many
times
  // we can keep track of our current surface or curve (entities with
multiple facets)
```

```
  // and only add_to_selection_buffer when we've determined our best
intersection
  // with that entity
  // another alternative is to just check the box_pick flag and when it is
true,
  // only call add_to_selection_buffer once per entity (we never have
intersection data for that anyway)




  typedef vtkstd::multimap<RefEntity*, DeferredSurfaceData> DeferredSurfaces;
  DeferredSurfaces deferred_surfaces;

  // loop over the picks that we got from our VTK picker
  for(iter = picks->begin(); iter != picks->end(); ++iter)
  {

      bool MultiUserWorkSpaceCheck(false);


    // get the geometry group for this entity
    vtkDataSet* ds = (*iter)->mData;
    GSurface* g_surface = NULL;
    GCurve* g_curve = NULL;
    SVPointerContainer::ArrayType* g_vertex_ptrs = NULL;

    vtkObject* backPtr = SVUtil::get_back_pointer(ds);
    g_surface = GSurface::SafeDownCast(backPtr);
    g_curve = GCurve::SafeDownCast(backPtr);
    g_vertex_ptrs =
      SVPointerContainer::ArrayType::SafeDownCast(ds->GetCellData()-
>GetArray("Pointers"));

    // if we don't have a geometry group, go to the next pick
    if(!g_surface && !g_curve && !g_vertex_ptrs)
      continue;

    // for each intersection with this geometry group
    for(jter = (*iter)->mPickedCells.begin(); jter != (*iter)-
>mPickedCells.end(); ++jter)
      {
      // get the point
      double point[4] = { (jter)->mPoint[0],
                         (jter)->mPoint[1],
                         (jter)->mPoint[2],
                         1.0};

      // transform to viewpoint to get depth
      ren->SetWorldPoint(point);
      ren->WorldToView();
      ren->GetViewPoint(point);

      // if we handle our pick from a surface
      if( g_surface && (filter_types[0] || filter_types[1] || filter_types[2]
|| filter_types[3]))
      {
        RefEntity* ref_face = g_surface->ref_entity();
        assert(ref_face != NULL);
        if(!ref_face)
          continue;

            //int PickedVolumeID(0);

            /*RefEntity* APickedVolumeID = g_surface->sample_parent()-
>ref_entity();
            if (APickedVolumeID)
            {
                  PickedVolumeID= APickedVolumeID;
            }*/


            //int PickedVolumeID = g_surface->


            //Multi-User Picked Volume ID
            //int PickedVolumeID=g_surface->top_entity()->get_source_model()-
>get_cgm()->id(ref_face);


                         // if we are picking volumes
                                if( filter_types[0] || filter_types[2] )
                              {
                                // find the volume that was picked
                                GEntity* volume = g_surface->sample_parent();
                                if(volume)
                                {
                                    PickedGeomEntity* pick =
PickedGeomEntity::New(volume->ref_entity(), g_surface->top_entity()-
>get_source_model()->get_cgm());
                                    if(box_pick &&
!MultiCellPicker::GetBoxAcceptPartialCells())
                                    {
                                      deferred_surfaces.insert(
DeferredSurfaces::value_type(ref_face, DeferredSurfaceData(pick, (jter)-
>mPoint, point[2], (jter)->mZeroTol)) );
                                    }
                                    else
                                    {
                                      //Put Multi-User Code Here
                                      RefEntity* MUpicked_entity = pick-
>ref_entity();
                          RefEntity* MUvolume = MUpicked_entity;
                                      int MUPickedVolumeID = pick-
>get_interface()->id(MUvolume);

                                      MultiUserWorkSpaceCheck =
MultiUserWorkSpaces(MUPickedVolumeID);
                                      if (MultiUserWorkSpaceCheck==true)
                                      {
                                        add_to_selection_buffer(pick, (jter)-
>mPoint, point[2] + gVolumeOffset, (jter)->mZeroTol);
                                        pick->Delete();
                                      }
                                    }
                                 }
                               }

                              // if we are picking surface
                              if( filter_types[0] || filter_types[3] )
                              {
                                PickedGeomEntity* pick =
PickedGeomEntity::New(ref_face, g_surface->top_entity()->get_source_model()-
>get_cgm());


                                //Put Multi-User Code Here
                                RefEntity* MUSpicked_entity = g_surface-
>sample_parent()->ref_entity();
                      RefEntity* MUSvolume = MUSpicked_entity;
                                int MUSPickedVolumeID = pick-
>get_interface()->id(MUSvolume);

                                MultiUserWorkSpaceCheck =
MultiUserWorkSpaces(MUSPickedVolumeID);

                      if (MultiUserWorkSpaceCheck==true)
                                {
                                      add_to_selection_buffer(pick, (jter)-
>mPoint, point[2] + gSurfaceOffset, (jter)->mZeroTol);
                                    pick->Delete();
                                }
                              }



                              // if we are picking bodies
                              if( filter_types[0] || filter_types[1] )
                              {
                                // find the body that was picked
                                // get the volume
                                GEntity* volume = g_surface->sample_parent();
                                RefEntity* body = NULL;
                                // then get the body from the volume
                                if(volume)
                                {
                                    body = volume->sample_parent() ? volume-
>sample_parent()->ref_entity() : NULL;
                                }

                                if(body)
                                {
                                    PickedGeomEntity* pick =
PickedGeomEntity::New(body, g_surface->top_entity()->get_source_model()-
>get_cgm());
                                    if(box_pick &&
!MultiCellPicker::GetBoxAcceptPartialCells())
                                    {
```

```
                                      deferred_surfaces.insert(
DeferredSurfaces::value_type
                                          (ref_face,
DeferredSurfaceData(pick, (jter)->mPoint, point[2], (jter)->mZeroTol)) );
                                    }
                                    else
                                    {
                                      //Put Multi-User Code Here
                                      add_to_selection_buffer(pick, (jter)-
>mPoint, point[2] + gBodyOffset, (jter)->mZeroTol);
                                      pick->Delete();
                                    }
                                }
                              }


      }

      // if we handle our pick from a curve
      if( g_curve && (filter_types[0] || filter_types[4]) )
      {
        RefEntity* ref_edge = g_curve->ref_entity();
        assert(ref_edge != NULL);
        if(!ref_edge)
          continue;

        // add this curve to our selection buffer
        PickedGeomEntity* pick = PickedGeomEntity::New(ref_edge, g_curve-
>top_entity()->get_source_model()->get_cgm());

            //Put Multi-User Code Here
        add_to_selection_buffer(pick, (jter)->mPoint, point[2] +
gCurveOffset, (jter)->mZeroTol);
        pick->Delete();

      }

      // if we handle our pick from a vertex
      if( g_vertex_ptrs && (filter_types[0] || filter_types[5]) )
      {
        // ask the vertex group what RefVertex this vtk point is
        GVertex* g_vertex =
SVPointerContainer::get_ptr<GVertex>(g_vertex_ptrs, jter->mId);
        RefEntity* ref_vertex = g_vertex->ref_entity();
        assert(ref_vertex != NULL);
        if(!ref_vertex)
          continue;

        // add it to our selection buffer
        PickedGeomEntity* pick = PickedGeomEntity::New(ref_vertex, g_vertex-
>top_entity()->get_source_model()->get_cgm());

            //Put Multi-User Code Here
        add_to_selection_buffer(pick, (jter)->mPoint, point[2] +
gVertexOffset, (jter)->mZeroTol);
        pick->Delete();
      }

    } // end for jter
  }   // end for iter

  // process deferred surfaces
  // these surfaces are deferred because we want entire volumes or entire
bodies to be
  // enclosed by a box pick
  DeferredSurfaces::iterator kter;
  for(kter = deferred_surfaces.begin(); kter != deferred_surfaces.end();
++kter)
  {
    PickedGeomEntity* pick = kter->second.mPickedGeom;
    DeferredSurfaceData* extra_data = &kter->second;
    RefEntity* picked_entity = pick->ref_entity();
    RefEntity* volume = picked_entity;
    if(volume)
    {
      DLIList<RefEntity*> child_surfaces;
      pick->get_interface()->get_child_ref_entities(volume, child_surfaces);

        //Multi-User
        int MUPickedVolumeID = pick->get_interface()->id(volume);


        int i;
      bool any_not_in_list = false;
      for(i=0; i<child_surfaces.size() && any_not_in_list != true; i++)
      {
        RefEntity* surface = child_surfaces.next(i);
        DeferredSurfaces::iterator zter = deferred_surfaces.find(surface);
        if(zter == deferred_surfaces.end())
        {
          any_not_in_list = true;
        }
            //add volume numbers
      }
      if(any_not_in_list == false)
      {
            //Add Multi-User Code Here
        add_to_selection_buffer(pick, extra_data->mPoint, extra_data->mZ +
gVolumeOffset, extra_data->mTol);
      }
      continue;
    }
    RefEntity* body = picked_entity;
    if(body)
    {
      DLIList<RefEntity*> child_volumes;
      DLIList<RefEntity*> child_surfaces;
        pick->get_interface()->get_child_ref_entities(body, child_volumes);
      //body->get_child_ref_entities(child_volumes);
      int i;
      for(i=0; i<child_volumes.size(); i++)
      {
        DLIList<RefEntity*> children;
        pick->get_interface()->get_child_ref_entities(child_volumes.next(i),
children);
        child_surfaces += children;
      }
      child_surfaces.uniquify_unordered();

      bool any_not_in_list = false;
      for(i=0; i<child_surfaces.size() && any_not_in_list != true; i++)
      {
        RefEntity* surface = child_surfaces.next(i);
        DeferredSurfaces::iterator zter = deferred_surfaces.find(surface);
        if(zter == deferred_surfaces.end())
        {
          any_not_in_list = true;
        }
      }
      if(any_not_in_list == false)
      {
            //Put Multi-User Code Here
        add_to_selection_buffer(pick, extra_data->mPoint, extra_data->mZ +
gBodyOffset, extra_data->mTol);
      }

    }
  }

  // cleanup our references to the picked entities
  for(kter = deferred_surfaces.begin(); kter != deferred_surfaces.end();
++kter)
  {
    kter->second.mPickedGeom->Delete();
  }

}

void GGeomPicker::process_groups()
{
  // get the selection buffer
  const EntitySelection* p_old_selections;
  int num_old_selections;
  get_selection_buffer(p_old_selections, num_old_selections);

  // if there is nothing, just return
  if(num_old_selections == 0)
    return;

  // copy selections
  vtkstd::vector<EntitySelection> old_selections;
  old_selections.insert(old_selections.end(), p_old_selections,
p_old_selections+num_old_selections);

  // clean out selection buffer
  clear_selection_buffer();

  vtkstd::vector<EntitySelection>::iterator iter;
  for(iter = old_selections.begin(); iter != old_selections.end(); ++iter)
  {
    PickedGeomEntity* geom_entity = dynamic_cast<PickedGeomEntity*>(iter-
>mEntity);
    if(!geom_entity)
      continue;
    RefEntity* entity = geom_entity->ref_entity();
    if(!entity)
      continue;
    DLIList<RefEntity*> groups;
    geom_entity->get_interface()->get_owning_groups(entity, groups);

    for(int i=0; i<groups.size(); i++)
    {
            PickedGeomEntity* pick =
PickedGeomEntity::New(groups.get_and_step(), geom_entity->get_interface());

        //Put Multi-User Code Here
      add_to_selection_buffer(pick, iter->mPickIntersection, iter->mZValue,
false);
      pick->Delete();
    }
  }
}

//Multi-User Workspace Function
bool MultiUserWorkSpaces(int PickedVolumeID)
{
 //Open Config File
  std::ifstream WorkSpaceFile;
  WorkSpaceFile.open("C:\\CubitConnect\\ConfigFiles\\Workspace.txt");


  std::vector<int> WorkSpaceNumbers;
  string dummy_string;

  while (WorkSpaceFile.eof()!=true)
  {
        std::getline(WorkSpaceFile, dummy_string);
        if(dummy_string.find("all")!=std::string::npos ||
dummy_string.find("ALL")!=std::string::npos)
        {
              return true;
        }
        int dummy_int= atoi(dummy_string.c_str());
        WorkSpaceNumbers.push_back(dummy_int);
  }
    //Check if PickedVolumeID is contained in the config file
  for (int i=0; i<WorkSpaceNumbers.size(); i++)
  {
        if (WorkSpaceNumbers[i]==PickedVolumeID)
        {
              return true;
        }
  }
return false;


}
```

APPENDIX B. CUBIT PEER TO PEER IMPLEMENTATION DOCUMENTATION

B.1 Introduction

Cubit is a mesh generation tool developed primarily by Sandia National Laboratories. Constant research is being done to upgrade the capabilities of Cubit at both BYU and Carnegie Mellon University.

Cubit comprises of two distinct portions:

1) Cubit Core (executes the FEA method)
2) Claro (Graphical User Interface)

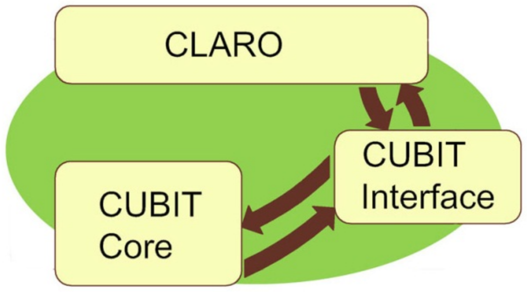


Figure B-0-1: CUBIT Process Hierarchy

The user interacts with Claro and whenever the user creates geometry, mesh, etc. a command string is automatically generated and passed through to the Cubit Core through the Cubit Interface. That command is then executed and the results are passed back to the GUI for displaying through the same interface.

Most of the programming was done in C++ while python was used to create some aspects of the GUI. Qt creator was the main graphics generation software used. The programming for the project was done using Visual Studio 2008. Setting up Cubit is explained in depth in a later section.

B.2 Problem Statement: Cubit Connect

Introduction:

Cubit is a finite element mesh generation software developed by Sandia National Laboratories. Cubit has a user friendly GUI than most of the meshing programs available. However, Cubit does not have solving capabilities. Cubit's source code is available for academic purposes and each student in the project will be required to get access to this through Sandia. Further information on this will be provided later on.

Objective:

Modify Cubit to be a multi-user program with the following capabilities by the end of this semester:

1) Three users can get on to the same preprocessed model.
2) The program places each cursor at a random location on the mesh.
3) Each user should be able to:
   a) Skew elements
   b) Apply loads
   c) Apply constraints around their locality.

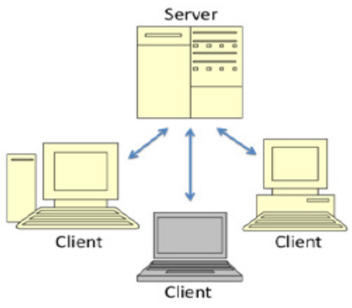


Figure B-0-2: CS Architecture

Project Distribution:

The group as a whole should develop a method to get 3 users on to the same model.

This will involve figuring out client server architecture and how to use that to enable the communication between three computers running Cubit.

Each member of the Cubit team:

1) Would be assigned one of the following tasks to figure out a way to :
   a) Enable users to skew elements
   b) Enable user to apply loads
   c) Enable users to apply constraints in the proposed multi-user environment

2) Should create relevant dialog boxes and programs that enable the assigned task.

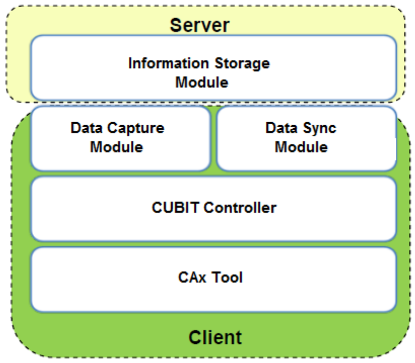


Figure B-0-3: CUBIT Connect Modules

After completing of the above tasks, the team should combine everything into one package and then demonstrate the combined capabilities of the new multi-user Cubit to the class by the end of winter semester 2011.

Deliverable:

A working software package that demonstrates multi-user capabilities of Cubit by the end of the semester.

Resources:

A similar program for Siemens NX called NX-Connect was developed by BYU students and the documentation is available for review.
A document for a Cubit multi-user architecture is being prepared and should be available soon.

B.3 Building Cubit on a Local Machine

Download the following software:

1) Swig-1.3.40 URL: http://sourceforge.net/projects/swig/files/swigwin/swigwin-1.3.40/
2) Qt for Open Source C++ development on Windows (VS2008)
   URL: http://qt.nokia.com/downloads/windows-cpp-vs2008
3) Cmake version 2.8 .4 Windows (Win32 Installer)
   URL: http://www.cmake.org/cmake/resources/software.html
4) Tortoise SVN version 1.6.9 URL: http://sourceforge.net/projects/tortoisesvn/files/Application/1.6.9/
5) Python 2.7 x86 msi installer URL: http://python.org/download/releases/2.7/
6) Microsoft Visual Studio 2008 (Express is alright)

Make sure to download the 32bit versions of each of the above software. This was built on a machine running MS Windows 7 Enterprise Edition with a Quad Core Xenon W3520 processor and 12GB of memory.

Download and install the software packages one at a time. If you are asked to restart the computer after an installation, please do so then continue with the next installation.

Make the following folders in your C: drive

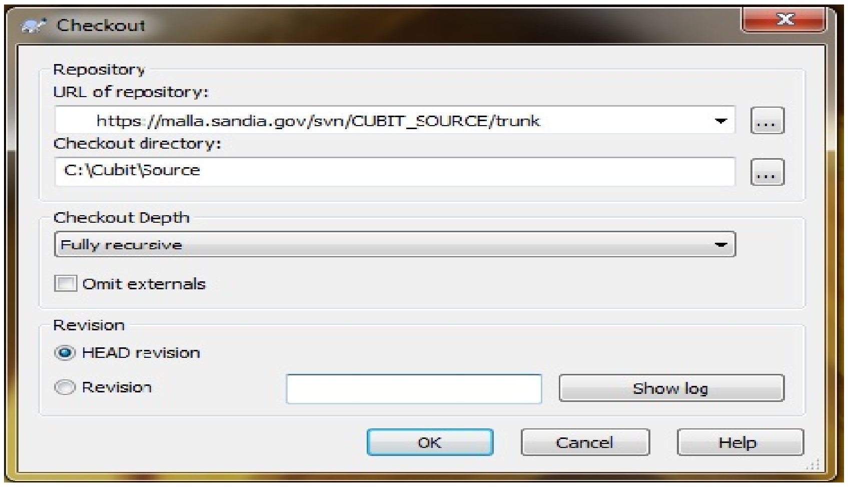
1) Cubit
2) Cubit\Build
3) Cubit\Source
4) windows_libs

Downloading Cubit Source:

Make sure you get individual access to the Cubit source code repository from Sandia. Specifically ask them to grant you access to the ACIS group and AcisTweakToolCAT file.

After getting access to the source code follow these steps:

1) After installing Tortoise SVN right click on the folder C:\Cubit\Source and select "SVN Checkout". Use the following settings:


Checkout

Repository

URL of repository: https://malla.sandia.gov/svn/CUBIT_SOURCE/trunk

Checkout directory: C:\Cubit\Source

Checkout Depth: Fully recursive

☐ Omit externals

Revision

● HEAD revision

○ Revision

Figure B-0-4: Tortoise SVN Checkout

URL of repository: http://malla.sandia.gov/svn/CUBIT_SOURCE/trunk

Checkout directory: C:\Cubit\Source

2) Leave everthing else in their default values and click OK. Enter the username and password provided to you by Sandia Labs.
3) Download should start immediately and will take about 1 hr, depending on your network speed, for the process to complete.

Downloading Windows Libraries for Cubit:

Right click on the folder named windows_libs (created earlier) and select "SVN Checkout". Use the following settings:

URL of repository: http://malla.sandia.gov/svn/LIBRARIES/windows_libs_2008/trunk/

Checkout directory: C:\windows_libs\

Leave everthing else in their default values and click OK. Enter the username and password provided to you by Sandia Labs.

Download should start immediately and will take about 1 hr, depending on your network speed, for the process to complete.

Changing Environment Variables:

1) Right click on Computer → Properties → Advanced system settings → Environment Variables
2) Under Sytem variables click on New
3) Add the following:
    i. Variable Name: CUBITROOT
       Variable Value: c:\window_libs
    ii. Variable Name: CUBITPATH
       Variable Value:
       %CUBITROOT%\bin;%CUBITROOT%\VTK\VTK-5.2.0\bin;%CUBITROOT%\acis\acis21.1\bin\NT_VC9_DLL;%CUBITROOT%\acis\acis21.1\bin\NT_VC9_DLLD;%CUBITROOT%\camal\camal5.3.1\lib\Windows
4) Edit the system "PATH" Variable to have the following value at the end:
    Variable Name: PATH
    Variable Value: C:\Program Files\TortoiseSVN\bin;%CUBITPATH%;C:\swigwin-1.3.40\Lib;C:\Python27\libs;C:\QtSC\bin;C:\Program Files (x86)\CMake 2.8\bin

    You should also make sure the paths for the following are included in the variable:
    i. Tortoise (bin directory)
    ii. Qt (bin directory)
    iii. CMake (bin directory)
    iv. Python (libs directory)

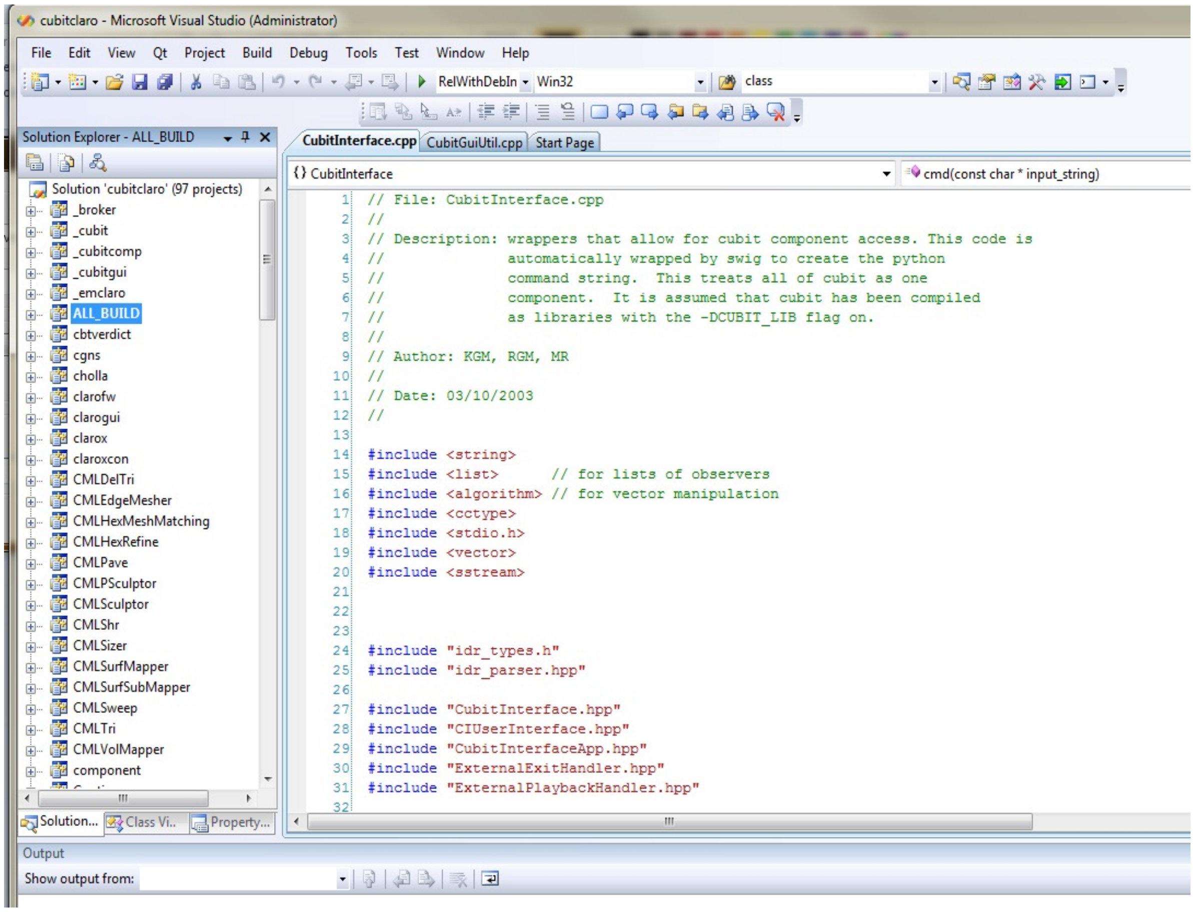
NOTE: Above variable values are example paths, therefore, make sure you enter the variable values as per the installation paths on your particular system.

Running CMake:

1) Open CMake(CMake-gui) under Start->All Programs->CMake
2) Select File -> Delete Cache to make sure that you start fresh and use the following values
3) Where is the source code: C:/Cubit/Source/trunk/cubitclaro
4) Where to build the binaries: C:/Cubit/Build
5) Make sure Advanced is checked on the top panel
6) Click once on Configure
7) A diaglog box will pop up asking the user to specify the generator for the project
8) Scroll down and select Visual Studio 9 2008 (Use default native compilers)
9) It is normal for CMake to give a warning about missing SVN file in the output window. Ignore it.
10) CMake will give you and error and ask for the python executable (these will be highlighted in Red).
11) In the middle window, under “name” scroll down to python execulatble and click on Value
12) Scroll to the directory of Python and select python.exe
13) Hit Configure again
14) CMake will give an error again and it will ask for the swig executable.
15) Click on the value of SWIG_EXECUTABLE and browse to the directory of and select swig.exe
16) Hit Configure again
17) CMake will now show some values in Red again.
18) Disregard those and hit Configure (If there is an error at this point. Delete Cubit\build folder, Open CMake and follow steps 1 therough 18 again.)
19) Once the configuration complete message comes up hit generate.
20) When the generation complete message comes up, close CMake

Building Cubit using Visual Studio 2008:

1) Open Visual Studio 2008
2) Click on file -> Open -> Project/Solution
3) Navigate to C:\Cubit\Build folder
4) Select the solution file "cubitclaro.sln"
5) This should populate the Solution Explorer window on the left with numerious projects related to Cubit
6)


cubitclaro - Microsoft Visual Studio (Administrator)
File Edit View Qt Project Build Debug Tools Test Window Help
RelWithDebIn
Win32
class
Solution Explorer - ALL_BUILD
CubitInterface.cpp
CubitGuiUtil.cpp
Start Page
{} CubitInterface
cmd(const char * input_string)
Solution...
Class Vi...
Property...
Output
Show output from:

Solution 'cubitclaro' (97 projects)
_broker
_cubit
_cubitcomp
_cubitgui
_emclaro
ALL_BUILD
cbtverdict
cgns
cholla
clarofw
clarogui
clarox
claroxcon
CMLDelTri
CMLEdgeMesher
CMLHexMeshMatching
CMLHexRefine
CMLPave
CMLPSculptor
CMLSculptor
CMLShr
CMLSizer
CMLSurfMapper
CMLSurfSubMapper
CMLSweep
CMLTri
CMLVolMapper
component

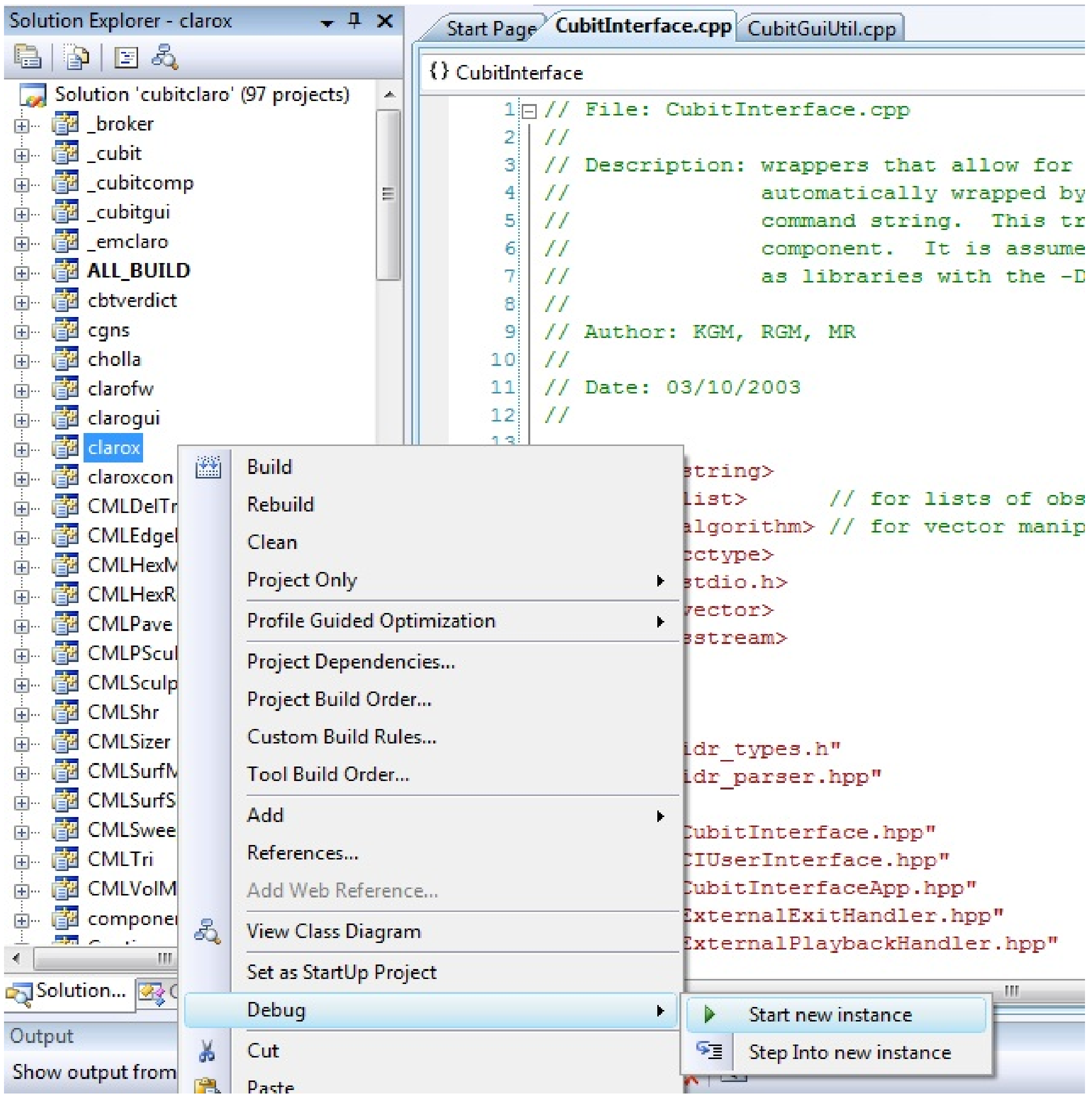
```
// File: CubitInterface.cpp
//
// Description: wrappers that allow for cubit component access. This code is
//              automatically wrapped by swig to create the python
//              command string.  This treats all of cubit as one
//              component.  It is assumed that cubit has been compiled
//              as libraries with the -DCUBIT_LIB flag on.
//
// Author: KGM, RGM, MR
//
// Date: 03/10/2003
//

#include <string>
#include <list>      // for lists of observers
#include <algorithm> // for vector manipulation
#include <cctype>
#include <stdio.h>
#include <vector>
#include <sstream>



#include "idr_types.h"
#include "idr_parser.hpp"

#include "CubitInterface.hpp"
#include "CIUserInterface.hpp"
#include "CubitInterfaceApp.hpp"
#include "ExternalExitHandler.hpp"
#include "ExternalPlaybackHandler.hpp"
```

Figure B-0-5: Snapshot of Cubit project in Visual Studio 2008

7) After everything is loaded click on Build on the top toolbar and click on Build Solution (or press F7)
8) This process would take a couple of hours to complete
9) Once the solution is built it should show 0 errors.
10) If a "microsoft incremental linker stopped working" error window pops up in Windows, just hit ignore and close the error window. Wait till the building is done in Visual Studio. The output window should show you about 4 errors should the above error window pops up. Simply click on Build Solution again and the errors will go away.

Running Cubit:

1. In the Solution Explorer find a project called “clarox”
2. Right click on the project and select Debug -> Start new instance
3. If a warning/information window pops up, just ignore it
4. Cubit should open up (disregard the missing documentation error popups inside Cubit)

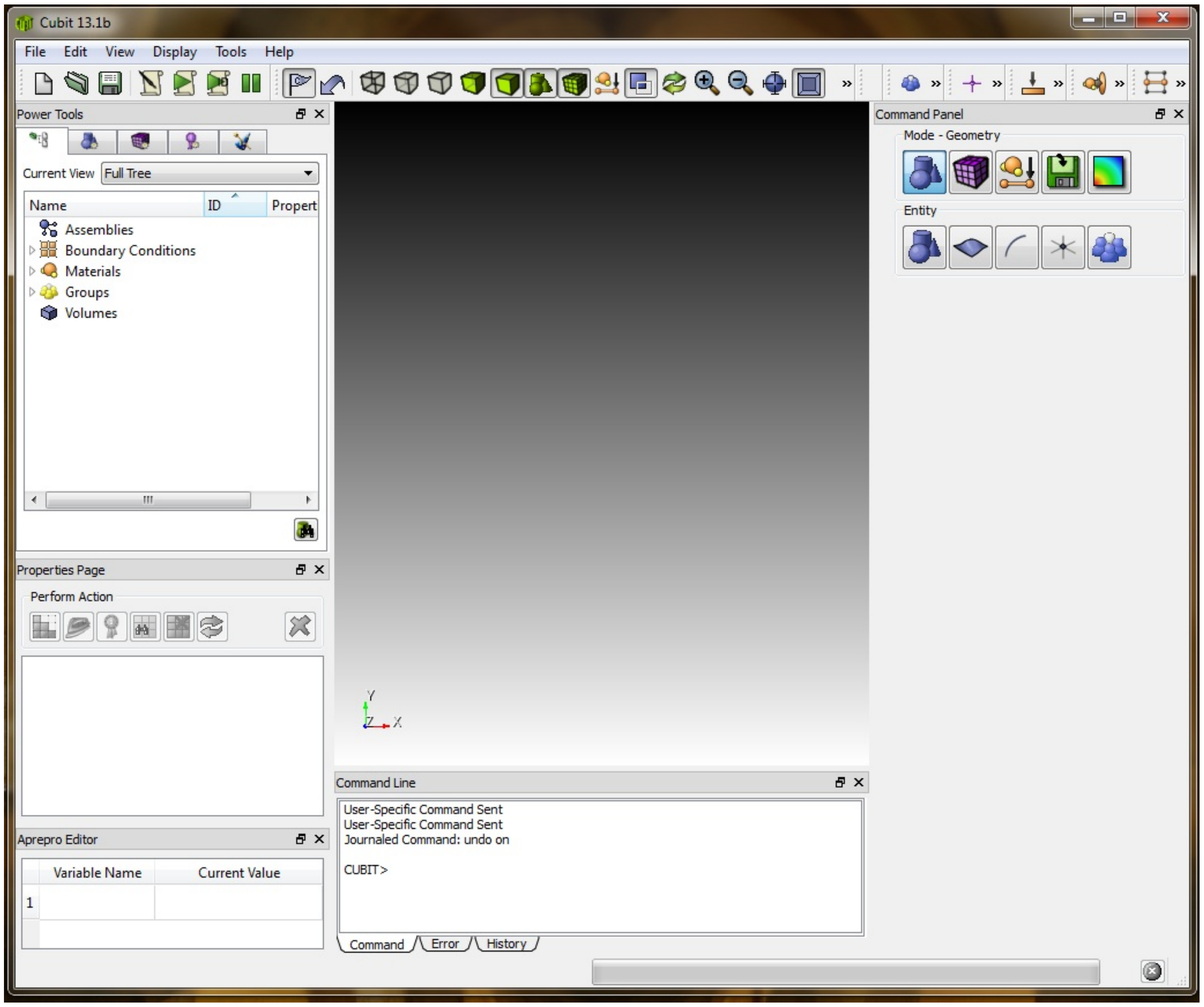


```
// File: CubitInterface.cpp
//
// Description: wrappers that allow for
//              automatically wrapped by
//              command string.  This tr
//              component.  It is assume
//              as libraries with the -D
//
// Author: KGM, RGM, MR
//
// Date: 03/10/2003
//
```

```
tring>
ist>      // for lists of obs
lgorithm> // for vector manip
ctype>
tdio.h>
ector>
stream>

dr_types.h"
dr_parser.hpp"

ubitInterface.hpp"
IUserInterface.hpp"
ubitInterfaceApp.hpp"
xternalExitHandler.hpp"
xternalPlaybackHandler.hpp"
```

Figure B-0-6: Running Cubit (GUI)

Cubit GUI should look something like this:



```
User-Specific Command Sent
User-Specific Command Sent
Journaled Command: undo on

CUBIT>
```

Figure B-0-7: Snapshot of the Cubit GUI Version 13.1b 32bit Version

Prepared by: Prasad Weerakoon prasadwee@yahoo.com

Contact Prasad if you have questions on setting up Cubit

For Cubit Support:

Email cubit-dev@sandia.gov or contact Dr. Karl Merkley or Mark Dewey at **Computational Simulation Software, LLC** on 801-717-2296 for

B.4 Documentation from Sandia (Reference)

Building CUBIT with Claro on Windows

Claro is CUBIT's graphical user interface environment developed by elemental technologies. This page contains instructions for setting up your machine for developing with Claro.

As a prerequisite, your machine should already be set up to build the command line version of Cubit.

0. Install Qt.

Qt is the C++ library that Claro uses to build its graphical user interface. It is a cross-platform library that allows us to use the same source code for any of our supported platforms. We use Qt 4.5 and greater. Follow the instructions for building and installing Qt for Visual Studio. Copying from another machine is not recommended, unless it is put in the same directory. The default pre-built Qt versions you can download from www.trolltech.com are built with mingw and are not compatible with Visual Studio.

1. Install Python

Claro is integrates with and uses a scripting language called python. Python is an object-oriented programming language, comparable to Perl, Tcl, Scheme, or Java. To compile Claro with Cubit will need to install python. Go to the following URL to download Python:

http://www.python.org/download/

The latest version of python is usually ok. As of this writing 2.6 works. Install it on your machine. For example: c:\python26

2. Install SWIG

SWIG is an interface compiler that generates wrappers to make C++ code available for use Python. Download swig from the following website:

http://www.swig.org/download.html

Swig downloads are maintained by Sourceforge. Unless you want to build your own executable from the source make sure you download the version that includes the executable. Here is the direct link to the download:

http://prdownloads.sourceforge.net/swig/swigwin-1.3.21.zip

Unzip and extract this file to a convenient location. For example: **C:\libs\swig**

3. Add the Python, SWIG and Qt to your PATH environment variable

The windows **Path** environment variable defines the default locations of executables on your computer. Add the directory you just installed python to, to your Path. Go to **start->settings->control panel->system**. Select the **Advanced** TAB and click on the **Environment Variables...** button. In the **System Variables** window scroll down to the **Path** variable. Select the **Edit...** button. Add the paths in there for your Qt installation, SWIG installation and Python installation. For example if Qt was installed in c:\Qt\4.5.3 and Python in c:\python26 and SWIG in c:\swig\swig-1.3.21, one would add “c:\Qt\4.5.3\bin;c:\python26;c:\swig\swig-1.3.21” in the dialog. Remember to use ; as the separator between each path. These paths help your programs find .dll files when they run, and also cmake looks at those paths to find .exe files for building Cubit.

4. Add PATH to Visual C++

Visual C++ will need to know the location of these .dll and .exe files. Open your copy of Visual Studio and go to the **Tools->Options** menu. Click on the **Directories** Tab. In this dialog, select the **executables** option and add at the end of the list $(PATH), if it is not already there. This means Visual Studio will pick up anything you set in your PATH for the system.

5. Download CubitClaro

Claro is Cubit's graphical user interface. It contains all the dialogs, widgets and their controllers to make the GUI go. Typical installation location is parallel to your Cubit main source directory. To checkout Claro from the Cubit source repository use Subversion. TotoiseSVN is an easy client to use and can be downloaded here. The path is:

https://malla.sandia.gov/svn/CUBIT_SOURCE/trunk/cubitclaro

6. Open CMake

Set CMake to get the source code from the cubitclaro folder downloaded from SVN. Choose a folder to build the binaries in and then press Configure. If any values show up in red after pressing Configure, try pressing it a few more times. Values can be set manually if CMake cannot assign them. Once all variable have turned gray, press ok to generate the build files.

7. Open the cubitclaro solution file

The previous step generates a Visual Studio solution file called cubitclaro.sln in the file where the binaries were built. Double click this to open it in Visual Studio.

8. Build the project

Use the **F7** key or select **Build->Build Solution** from the menu. After it has built, run the executable. If it displays an error about a missing .dll file, locate it and add the path to the Path environment variable. One file called msvcrtd.dll may need to be added to C:\WINDOWS\System32. If all else goes well, Cubit should be ready to use.

For further help with building Cubit, e-mail the Cubit-Dev mailing list with a description of the problem.

B.5 Transforming Cubit into a Multi-User Program

As discussed earlier, Cubit has two distinct components; the GUI and the Core. The first step in making Cubit multi-user is to capture the command strings that are passed to the Core from the GUI before they get sent to the Core. Then a networking program can be used to distribute these changes to the peer computers.

Preliminary Plan:

- Study Cubit source code (relevant areas) in depth.
- Develop a method to intercept the command strings before they are processed.
- List all the commands that are not model relevant (Change of view, rotate, etc.).
- Develop a networking program that sends command strings within the network.
- Integrate network program with Cubit to send/receive command strings

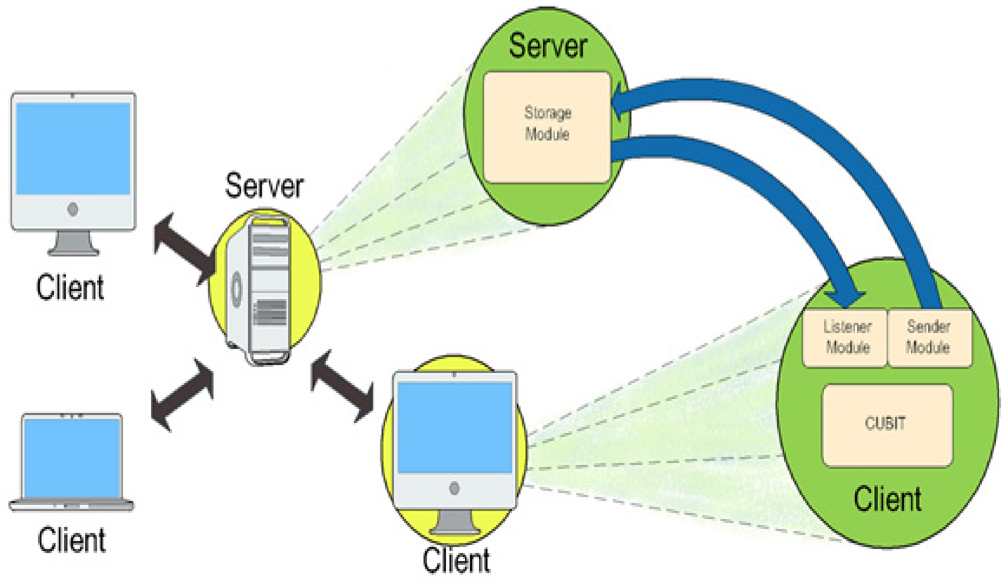


Figure B-0-8: Preliminary Cubit-Connect Architecture

Due to time constraints, it was decided to use a peer-to-peer(P2P) networking program rather than a Client-Server architecture to continue on with the project. A P2P networking client that was built with Qt Creator was modified for this purpose. Since the team ran in to difficulties integrating the networking program code inside of Visual Studio, an external networking program was used which ran parallel with Cubit.

Implemented Method:

1. Install Cubit along with the required software
2. Intercept all command strings passed on by the GUI before it is sent to the processor (Cubit core) -tweak Cubit Code
3. Filter out the command strings that do not have any bearing to the overall model (Eg: Change of view, zoom, etc.) and only pass the commands that are integral to the model to all users(peer-relevant commands) –Tweak Cubit Code
4. Write the peer relevant commands to a file that is being read by the Network Client
5. Pass the contents on the file to other users in the network
6. Write commands coming from peer computers to a file
7. Once user types the command for accepting updates from other users, read the file and send the commands to Cubit Core for execution
8. GUI updates the model

CUBIT-CONNECT ARCHITECTURE

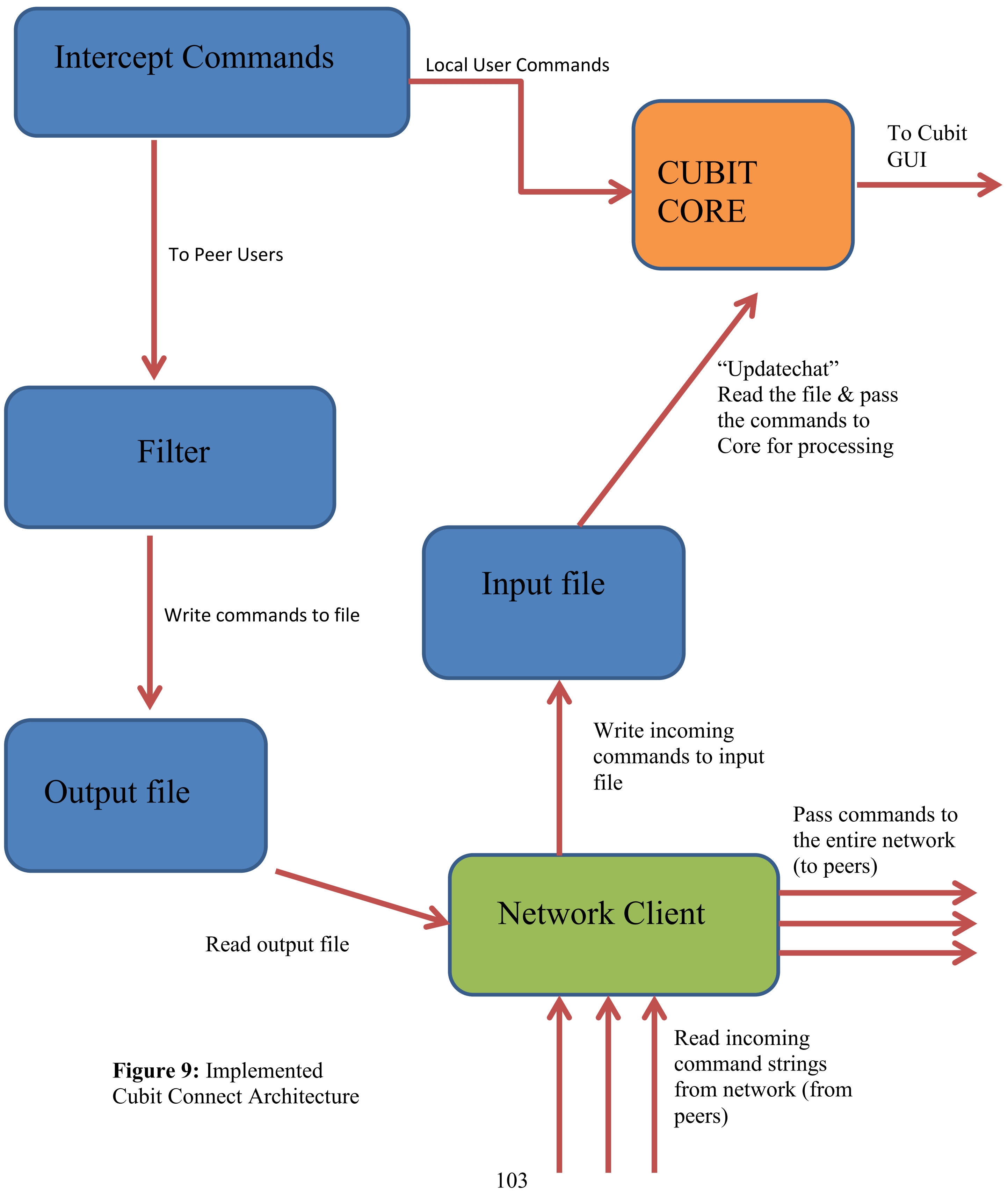

Intercept Commands
Local User Commands
CUBIT CORE
To Cubit GUI
To Peer Users
Filter
"Updatechat" Read the file & pass the commands to Core for processing
Write commands to file
Input file
Output file
Write incoming commands to input file
Pass commands to the entire network (to peers)
Network Client
Read output file
Read incoming command strings from network (from peers)

Figure 9: Implemented Cubit Connect Architecture

Modifying Cubit Source Code

For the proposed multi-user method to work, it was vital to figure out the place in the source code where the GUI sends commands to be executed to the core. Dr. Karl Merkley, who is a developer of Cubit, pointed out that CubitInterface.cpp file is where this occurred. In the function `CubitInterface::cmd(const char *input_string)` the command string is passed on to the Core for execution and the following is the original code from Sandia:

```
// Sends a command directly to cubit

void CubitInterface::cmd(const char *input_string)
{
  CIUserInterface *ui = CIUserInterface::instance();

  if(ui && ui->get_playback_handler())
    ui->parse_input_line(input_string);

  else
    UserInterface::instance()->parse_input_line_and_files(input_string);

  if(!gsObserver) return;


std::list<CubitInterface::CIObserve*>::iterator iter = gsObserver-
>mGUIObserver.begin();

  for (; iter != gsObserver->mGUIObserver.end(); iter++)
  {
    (*iter)->notify_command_complete();
  }

}
```

This file is located in:

Source……./cubitclaro/cubitcomp/cubit/app/CubitInterface.cpp

The modified code can be found on the next page. Note that it only reads and writes files to receive and send commands from peer computers. The network chat client runs parallel to Cubit and captures the commands written to the MultiuserTo.txt and propagates them through the network. Likewise, the network client writes the commands that were received from peer computer to the text file MultiuserFrom.txt. This file in then read by Cubit upon typing in the command “updatechat” and all the commands are then executed on the local computer.

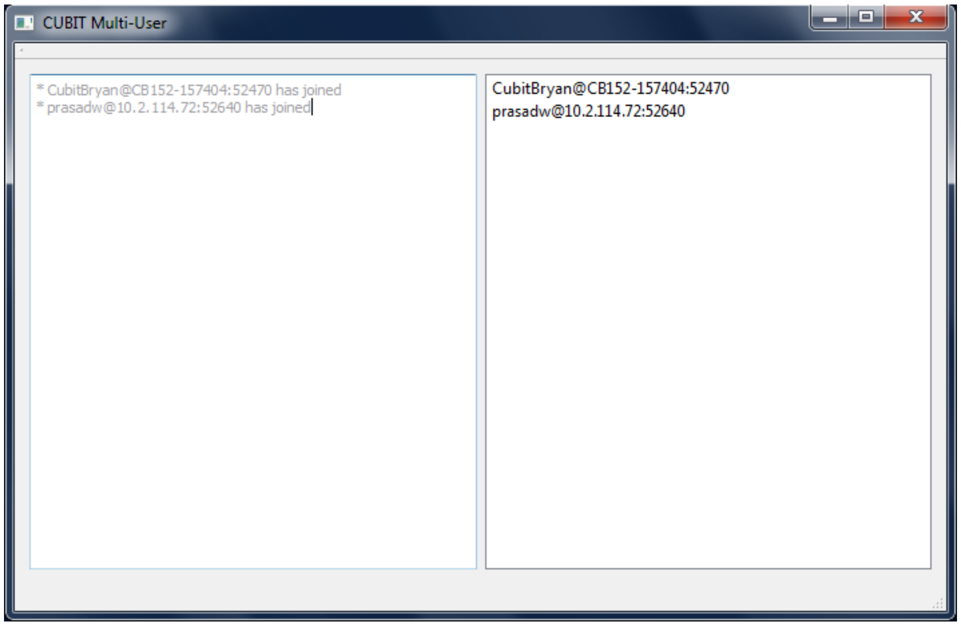
```
//Reading from the file
static int last_pos=0;
static fstream fout("C:\\myChat\\myChat\\MultiuserTo.txt", std::ios::in |
std::ios::out | std::ios::ate);

// Sends a command directy to cubit
void CubitInterface::cmd(const char *input_string)
{

    //Check to see if the command is user-specific
      string no_cmd[] =
{"graphics","draw","list","display","quality","preview","visibility", "null",
"color", "updatechat", "undo", "reset","open","import"};    //list of user
specific commands
      int NO_CMD_LEN = sizeof(no_cmd)/sizeof(string);  //# of user-specific
commands so far

      //See if the input_string contains a user-specific command
      bool toSend = true;
      string in_str(input_string);
      for(int i=0;i<NO_CMD_LEN;i++)
      {
            if(in_str.find(no_cmd[i]) != string::npos)
            {
              toSend = false;
            }

      }

      try
      {

            if (toSend)
            {
                  PRINT_INFO("Multi User Command Sent  \n ");

                  //Write to the output file (cubitin.cmds)
                  fout<<input_string<<std::endl;

            }
            if (!toSend) PRINT_INFO("User-Specific Command Sent \n");
            //if it is not user specific, send to other people on "chat"

      }
      catch (std::exception& e)
      {
            const char* name1;
            name1= e.what();
            PRINT_INFO (name1);
            PRINT_INFO ("Exception \n");
      }


      //to receive updates from other users
      ifstream fin;
      if(in_str.find("updatechat") != string::npos)
```

```
	{
		PRINT_INFO("Updating commands from other users \n");

		vector <string> input_string_in ;
		fin.open("C:\\myChat\\myChat\\MultiuserFrom.txt");
		if(!fin.is_open())
		{
			PRINT_INFO("Unable to open file \n");
		}else{
			PRINT_INFO("The file is open for reading");
		}

		//Keeps track of the position of the input file
		string dummy_string;
		int curr_pos=0;
		while (fin.eof()!=true)
		{
			getline(fin,dummy_string);
			if(!dummy_string.empty() && dummy_string.compare("")!=0)
curr_pos++;
			if(curr_pos>last_pos)
			{
				input_string_in.push_back(dummy_string);
			}
		}
		last_pos=curr_pos;

		//If inputfile in not empty, send the command string to Cubit
Core for execution
		if(!input_string_in.empty())
		{
			CIUserInterface *ui = CIUserInterface::instance();
			for (int i=0; i<input_string_in.size(); i++)
			{
				if(ui && ui->get_playback_handler())
				{
					ui-
>parse_input_line(input_string_in[i].c_str());
				}
				else
				{
					UserInterface::instance()-
>parse_input_line_and_files(input_string_in[i].c_str());

				}
				if(!gsObserver) return;

				std::list<CubitInterface::CIObserve*>::iterator iter
= gsObserver->mGUIObserver.begin();
				for (; iter != gsObserver->mGUIObserver.end();
iter++)
				{
					(*iter)->notify_command_complete();
				}
			}
		}else{
			PRINT_INFO("Nothing in the file \n");
```

```
            }

            fin.close();

      }else{

            CIUserInterface *ui = CIUserInterface::instance();
            if(ui && ui->get_playback_handler())
            {
                  ui->parse_input_line(input_string);
            }
            else
            {
                  UserInterface::instance()-
>parse_input_line_and_files(input_string);

            }
            if(!gsObserver) return;

            std::list<CubitInterface::CIObserve*>::iterator iter =
gsObserver->mGUIObserver.begin();
            for (; iter != gsObserver->mGUIObserver.end(); iter++)
            {
                  (*iter)->notify_command_complete();
            }
      }
}
```

B.6 Cubit Sample Tutorial

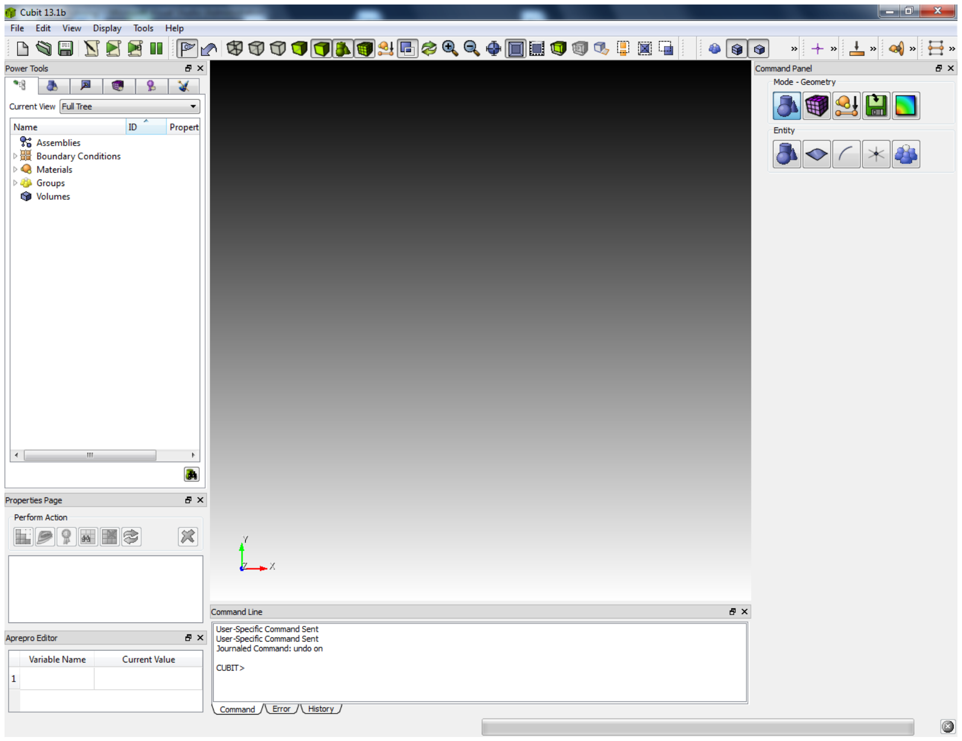
1. Start up the network program on all machines. They will automatically connect to each other on the LAN.



```
* CubitBryan@CB152-157404:52470 has joined
* prasadw@10.2.114.72:52640 has joined
```

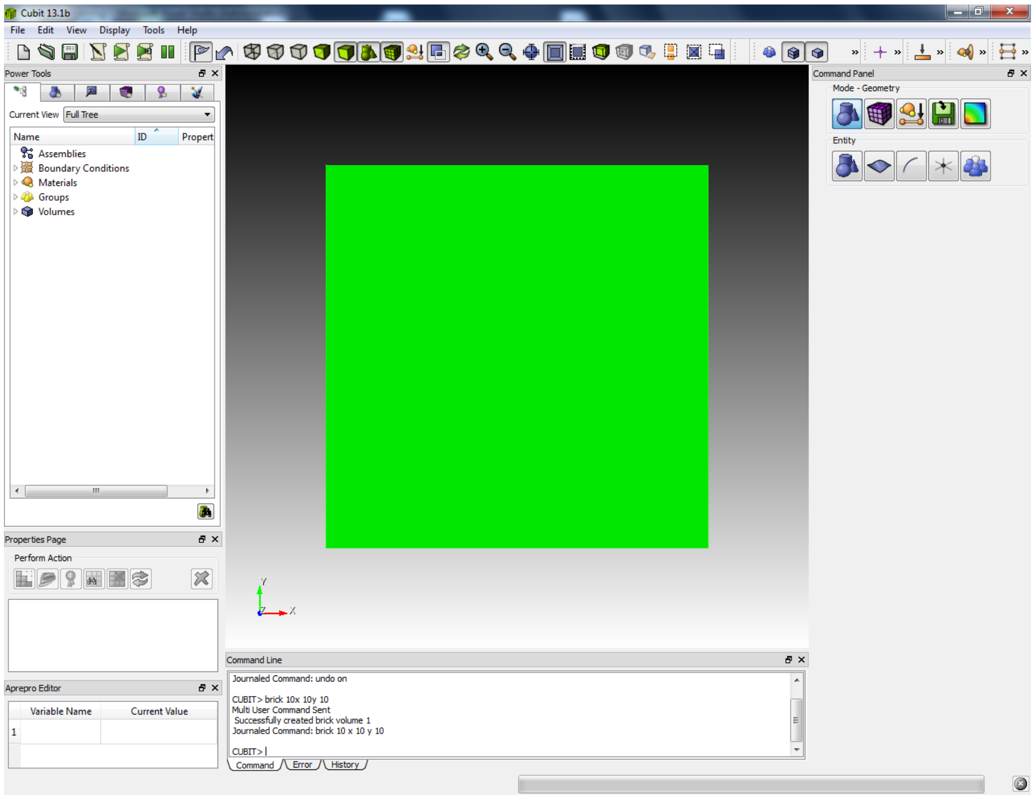
```
CubitBryan@CB152-157404:52470
prasadw@10.2.114.72:52640
```

2. Start CUBIT on each machine.



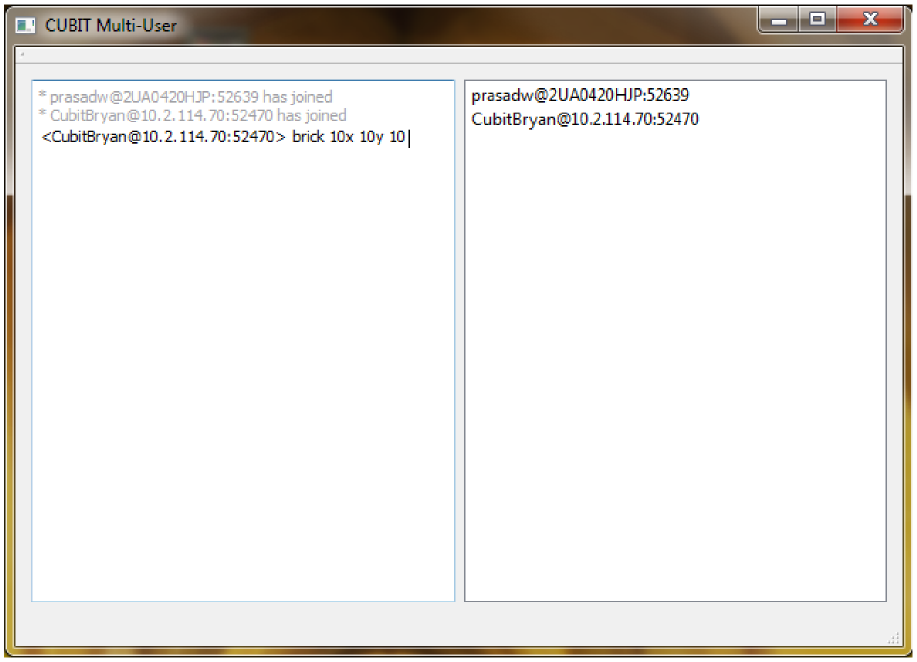
```
User-Specific Command Sent
User-Specific Command Sent
Journaled Command: undo on

CUBIT>
```

3. In CUBIT, on one of the machines, create a brick (brick 10x 10y 10).



```
Journaled Command: undo on

CUBIT> brick 10x 10y 10
Multi User Command Sent
 Successfully created brick volume 1
Journaled Command: brick 10 x 10 y 10

CUBIT> |
```

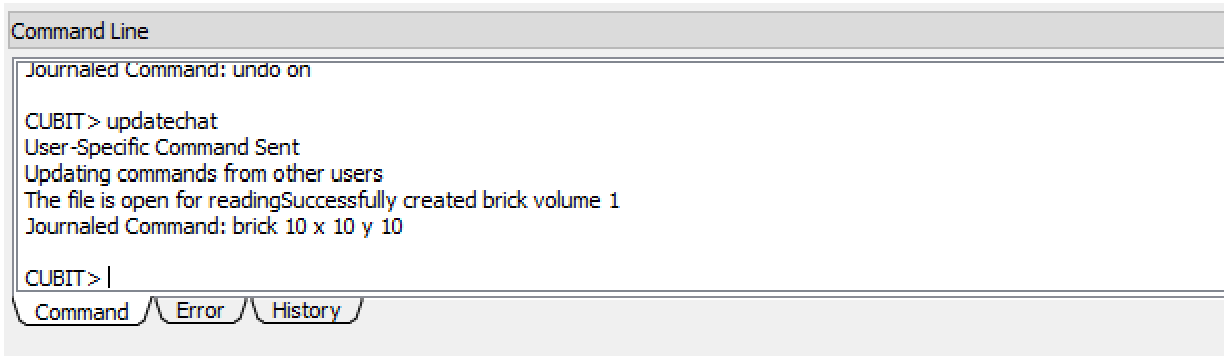
4. A message will shortly be sent to the network program.



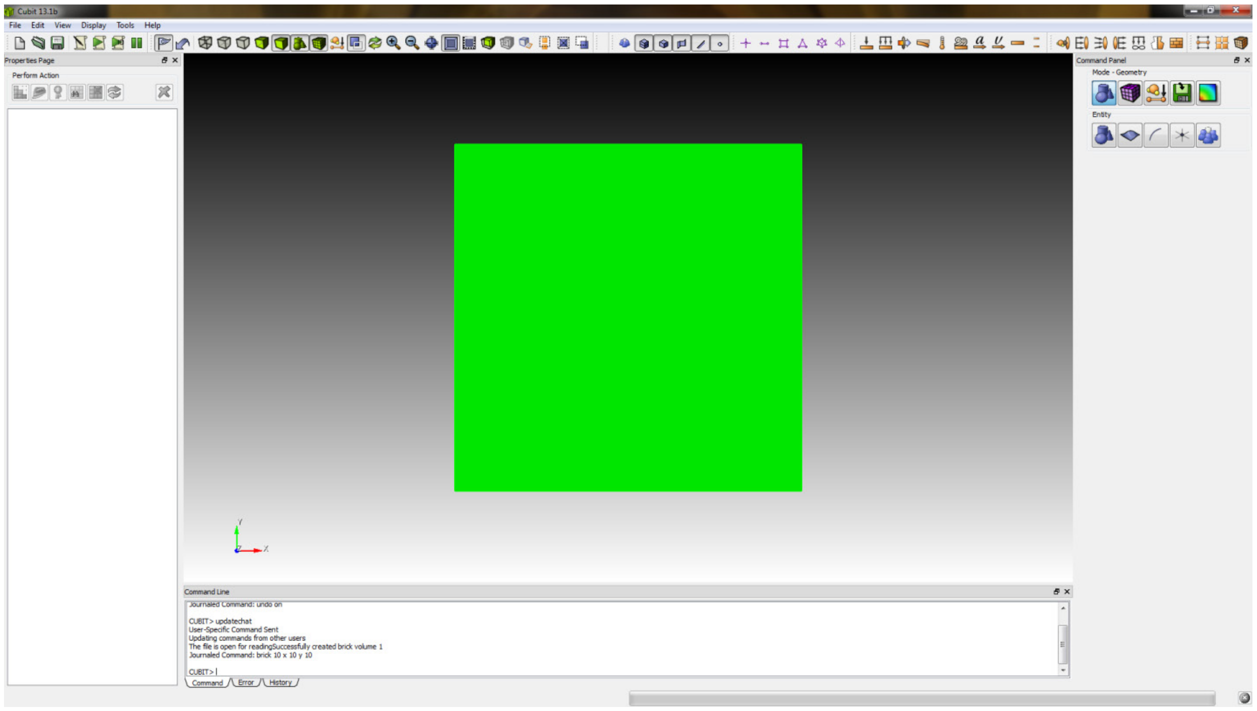
```
* prasadw@2UA0420HJP:52639 has joined
* CubitBryan@10.2.114.70:52470 has joined
<CubitBryan@10.2.114.70:52470> brick 10x 10y 10
```

```
prasadw@2UA0420HJP:52639
CubitBryan@10.2.114.70:52470
```

5. Once the network program has received the message, the other users can type ‘updatechat’ into CUBIT’s command prompt to get the changes made from other users.



```
Journaled Command: undo on

CUBIT> updatechat
User-Specific Command Sent
Updating commands from other users
The file is open for readingSuccessfully created brick volume 1
Journaled Command: brick 10 x 10 y 10

CUBIT> |
```



B. 7 Conclusions:

- Cubit Connect produces the results as expected
- Having access to the source code was easier to program as opposed to using API calls.
- For an FEA multi-user program which is hardware intensive, having an update command will be beneficial because if peer commands get automatically updated while the local user is working, it might freeze the program for a substantial amount of time while the commands get executed.
- A multi-user environment will revolutionize traditional collaborative engineering

B.8 Recommendations:

- Compile a comprehensive list of commands to be filtered
- Client-Server architecture rather than a peer-to-peer (P2P) architecture
- Embed the networking program inside Cubit code
- Use a database/lists rather than text files to propagate commands
- Think about how undo commands would work
- Importance of the update command
- Conflict resolution/decomposition of a model

APPENDIX C. CUBIT CLIENT-SERVER AND GUI DRIVEN IMPLEMENTATION V3.0

C.1 Introduction

CUBIT Connect is multi-user FEA pre-processing software developed collaborative by BYU and Sandia National Laboratory. After extensive research effort, unique and robust network architecture was developed for the CUBIT Connect environment. The current implementation utilizes both named pipes clients and TCP/IP. On the local level (in this case, on a local computer), CUBIT program is connected to a C# client using named pipes. This separation is due to the previous difficulty integrating C# programs, which is written in a managed environment, into the C++ CUBIT source code, which is written in an unmanaged environment. Once inter-process communication on the local level is established, the WAN communication can be easily established using TCP/IP sockets. The basic architecture is displayed in Figure 3.

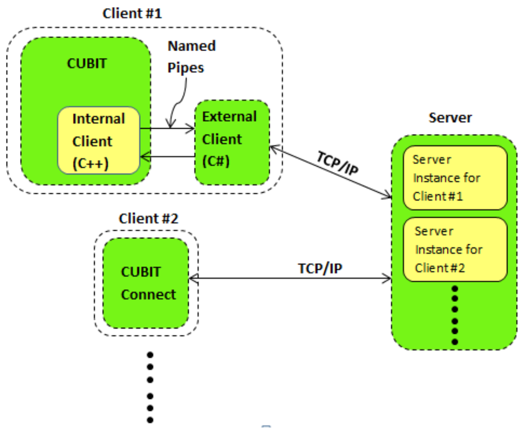


Figure C-0-1: CUBIT Connect v2.0 Architecture Utilizing Both Named Pipes and TCP/IP Clients.

Another research area that is going on parallel with the network architecture is model decomposition. In a multi-user environment, it is necessary to assign different areas (workspaces) for different users to work. Causally assigning a workspace to a user may not be effective, because it is likely for a user to accidently cross into another user's workspace. The result may end up creating chaos and would make the multi-user environment less productive. Therefore, users should be restricted to their own workspaces and, while viewing changes to the entire model, they are not allowed to make changes to others' workspaces.

From the solid foundation developed by previous researches, the next step for CUBIT Connect is to develop an efficient user Interface to combine these separate ideas into one. This task is consolidated into a ME 578 project, and the detail implementations are explained below.

C.2 Problem Statement: Cubit Connect V3.0

Introduction:

Cubit is finite element mesh generation software developed by Sandia National Laboratories. Cubit has a user friendly GUI than most of the meshing programs available. However, Cubit does not have solving capabilities. Cubit's source code is available for academic purposes and each student in the project will be required to get access to this through Sandia. Further information on this will be provided later on.

Project Description

Modify Cubit Connect prototype allow area/region and feature decomposition with the following capabilities by the end of this semester:

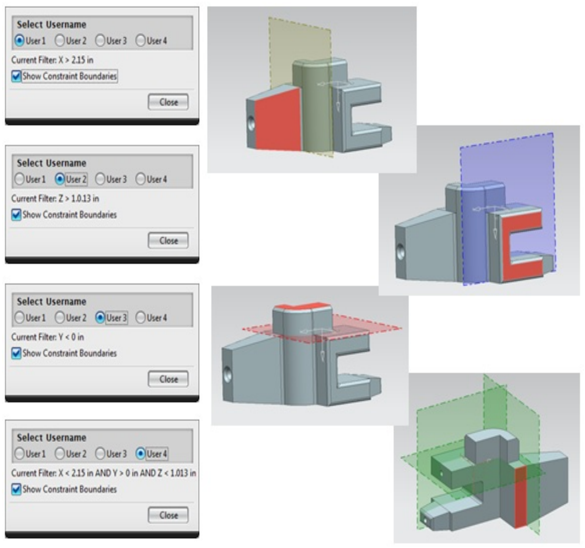
Modify/enhance current CUBIT code that assigns workspaces by restricting users to certain pre-assigned volume IDs.

Add a GUI for users to log on to the server where the server sends the pre-assigned user workspace data to the client computer automatically. This should be so that there is no hardcoding on the client computers.

Link this GUI to a database on the server that contains all the user data (user name, password, areas assigned, current work part, etc.)

Within the existing Cubit Connect architecture develop a method for partitioning the model with planes, bounding blocks and/cylinders, features, entity IDs, etc. so that multiple people can work without affecting or being affected by the work of others in the model. Provide a management control interface for setting up, transfer, and release of the partitioning features.

In the context of an assembly analysis provide locking capabilities with respect to components and partitioning within these features.


Select Username
[✓ User1] User2 User3 User4
Current Filter: X > 2.15 in
✓ Show Constraint Boundaries
Close

Select Username
User1 [✓ User2] User3 User4
Current Filter: Z > 1.0.13 in
✓ Show Constraint Boundaries
Close

Select Username
User1 User2 [✓ User3] User4
Current Filter: Y < 0 in
✓ Show Constraint Boundaries
Close

Select Username
User1 User2 User3 [✓ User4]
Current Filter: X < 2.15 in AND Y > 0 in AND Z < 1.013 in
✓ Show Constraint Boundaries
Close

Figure C-0-2: Part Decomposition in NX

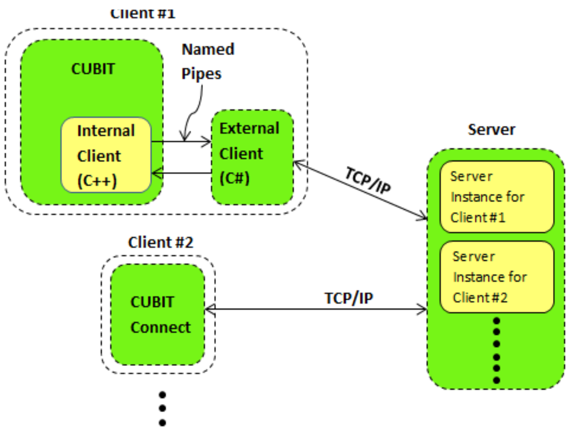


Figure C-0-3: CUBIT Connect Architecture

Deliverable

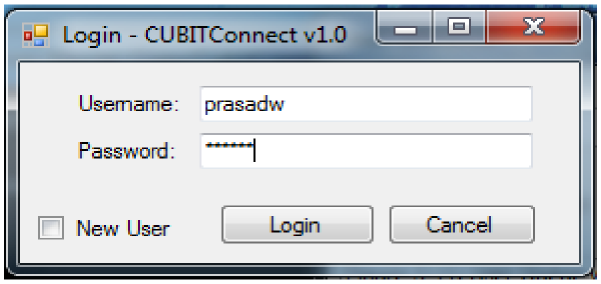
Cubit Connect with decomposition with an accompanying log on GUI and database.

Resources

Felicia Marshall's Thesis and work underway by Rob Moncur and Prasad Weerakoon.

C.3 GUI Implementation

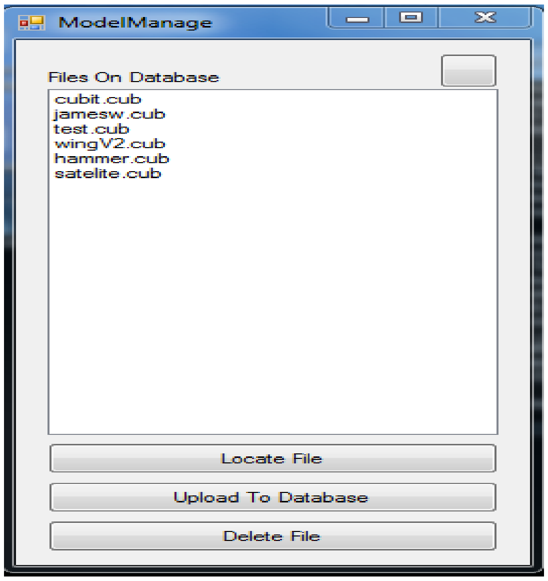
A series of GUIs were added to CUBIT Connect to handle and streamline workspace assignment. Figure 4-12 shows the User Login GUI (ULGUI) that checks user credentials and allows the user to proceed to select a model or part to work on. This also, checks what level of user (administrator, normal user, etc.) the current user is with the information stored in the server database. Furthermore, new users have the ability to add their information to the database using this GUI.



Username: prasadw
Password: ******

Figure C-0-4: **User Login GUI**

After the user successfully logs in, Model Management GUI (MMGUI) pops up (Figure 4-13). The MMGUI then queries the server for a list of models/parts currently stored on the server and displays them. The user can then locate the file, they would like to work on, on their local computer and open it. If the user is an administrator, the Admin Workspace Manager (AWM) GUI comes up; This GUI lets the administrator to assign workspaces to different users (Figure 4-14). The admin has to input the username of the user there are assigning the workspaces to and also, the volume ID numbers which that user has access to. The admin has the option to check the workspaces they just assigned by highlighting those in the model using "Check Workspace" button. The "Set Workspace" button sends the user and workspace information to the server.



Files On Database

cubit.cub
jamesw.cub
test.cub
wingV2.cub
hammer.cub
satelite.cub

Figure C-0-5: **Multiple Project/Model Management GUI**

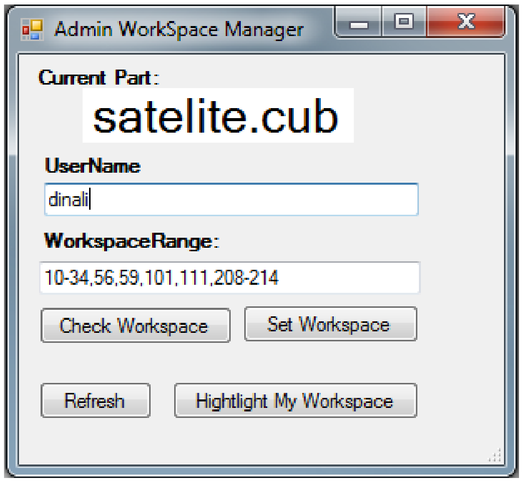


Figure C-0-6: **Workspace GUI for Administrator**

Software/Hardware Requirements

CUBIT Connect v3.0 is currently available on the following platforms:

☐ Windows 2000/XP/Vista/7, 32 and 64 bit

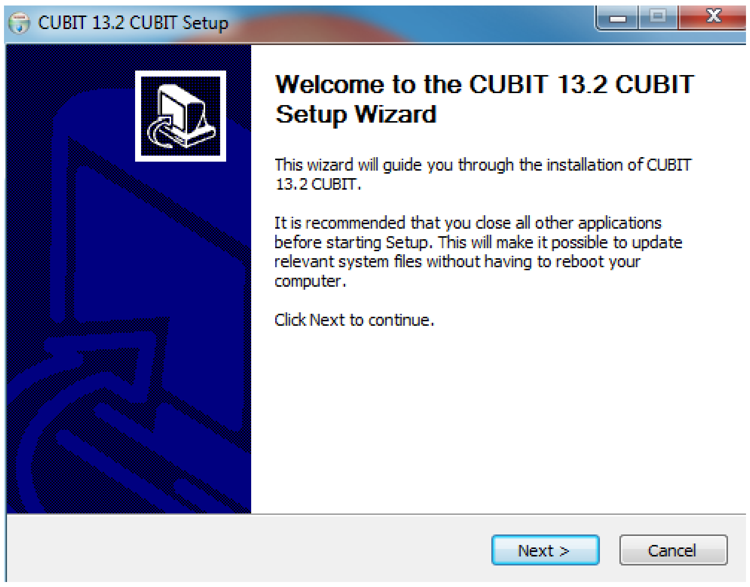
The Graphical User Interface version is available on all platforms.
For best results, local displays supporting OpenGL 1.5 is recommended.

C.4 Installation Instructions

Setting up Client Computers

Since the new network scheme is built directly inside of CUBIT Connect, the Client application is very easy to set up. The entire program is packaged into the CUBIT executable, and the installation process is almost identical to a regular CUBIT installation process. The detail steps are illustrated below:

1. Run Cubit-13.2-Win32.exe from CUBIT Connect V3.0 CD and a setup wizard should show up as displayed below:


CUBIT 13.2 CUBIT Setup

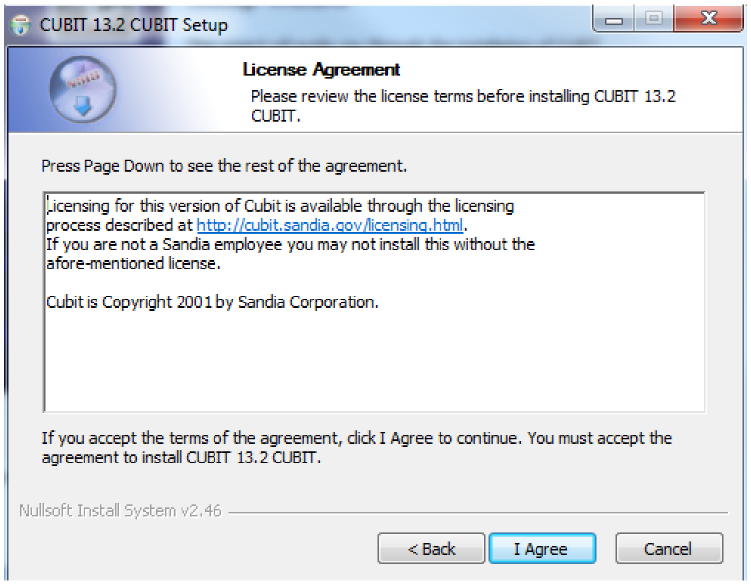
Welcome to the CUBIT 13.2 CUBIT Setup Wizard

This wizard will guide you through the installation of CUBIT 13.2 CUBIT.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

2. Click "Next >"


CUBIT 13.2 CUBIT Setup

License Agreement

Please review the license terms before installing CUBIT 13.2 CUBIT.

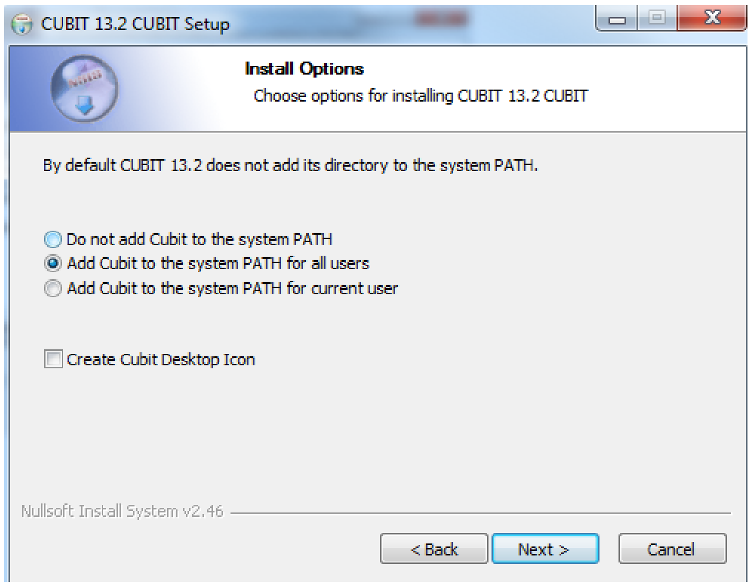
Press Page Down to see the rest of the agreement.

Licensing for this version of Cubit is available through the licensing process described at http://cubit.sandia.gov/licensing.html.
If you are not a Sandia employee you may not install this without the afore-mentioned license.

Cubit is Copyright 2001 by Sandia Corporation.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install CUBIT 13.2 CUBIT.

3. Click "I Agree"


CUBIT 13.2 CUBIT Setup

Install Options

Choose options for installing CUBIT 13.2 CUBIT

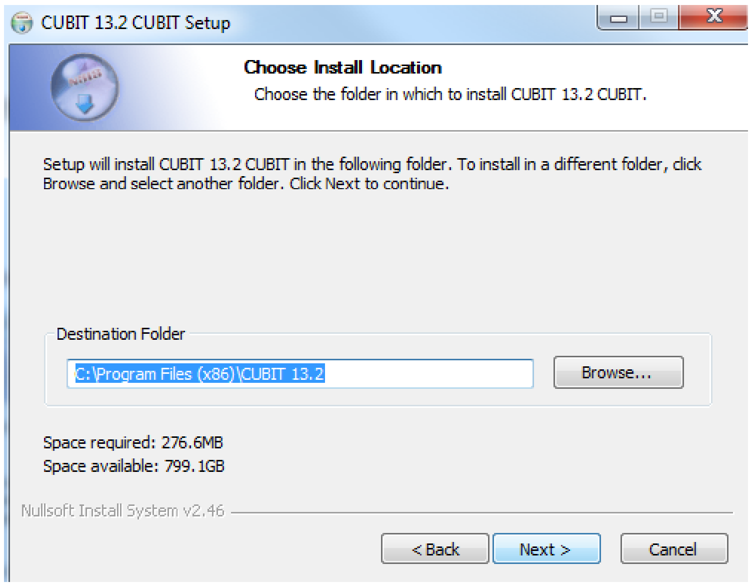
By default CUBIT 13.2 does not add its directory to the system PATH.

- Do not add Cubit to the system PATH
- [✓ Add Cubit to the system PATH for all users]
- Add Cubit to the system PATH for current user

☐ Create Cubit Desktop Icon

Nullsoft Install System v2.46

4. Select "Add Cubit to the system PATH for all users" and click "Next >"


CUBIT 13.2 CUBIT Setup

Choose Install Location

Choose the folder in which to install CUBIT 13.2 CUBIT.

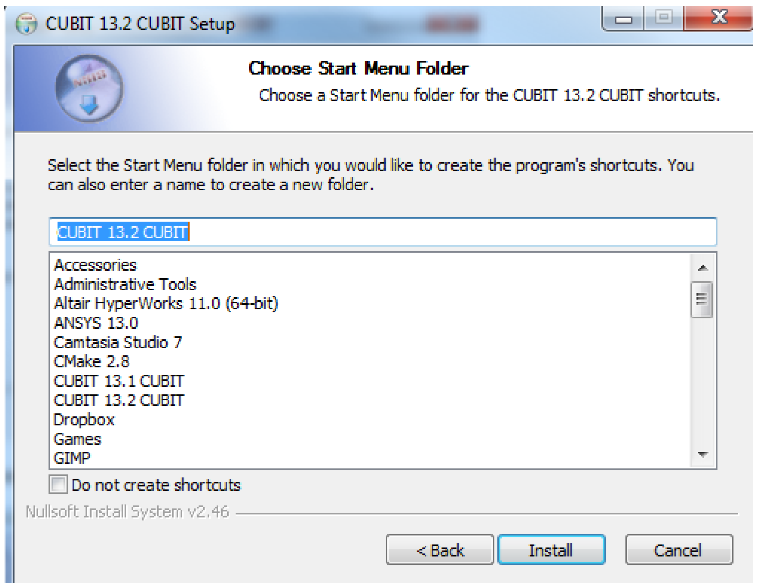
Setup will install CUBIT 13.2 CUBIT in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files (x86)\CUBIT 13.2

Space required: 276.6MB
Space available: 799.1GB

Nullsoft Install System v2.46

5. Click "Next >"



Choose Start Menu Folder

Choose a Start Menu folder for the CUBIT 13.2 CUBIT shortcuts.

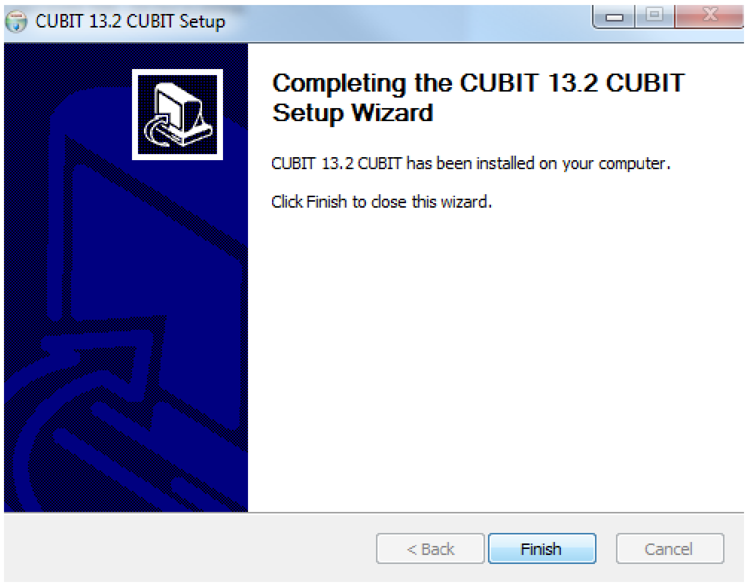
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

CUBIT 13.2 CUBIT

Accessories
Administrative Tools
Altair HyperWorks 11.0 (64-bit)
ANSYS 13.0
Camtasia Studio 7
CMake 2.8
CUBIT 13.1 CUBIT
CUBIT 13.2 CUBIT
Dropbox
Games
GIMP

Do not create shortcuts

6. Click "Install"

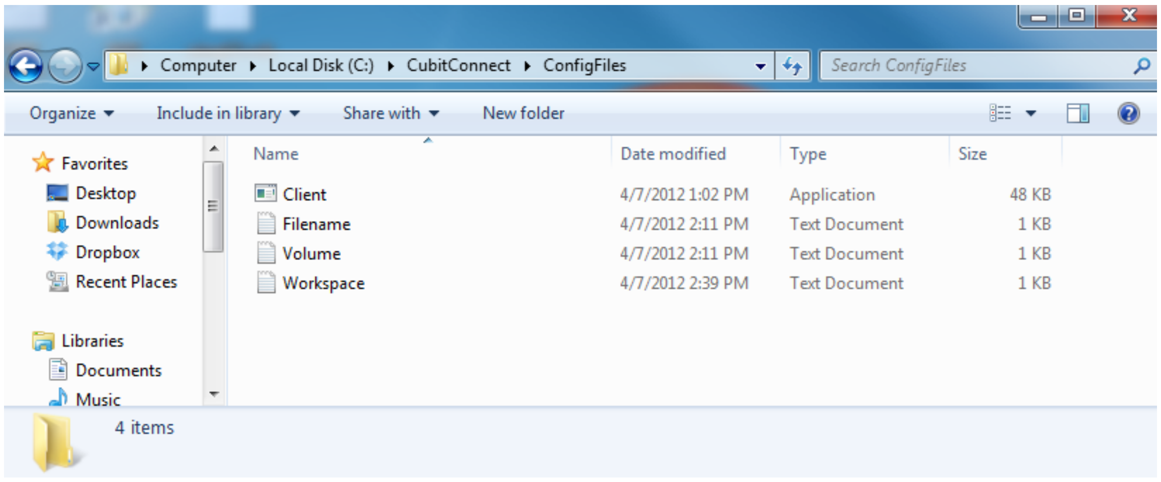


Completing the CUBIT 13.2 CUBIT Setup Wizard

CUBIT 13.2 CUBIT has been installed on your computer.

Click Finish to close this wizard.

7. Click "Finish"
8. Create a folder in C drive called "CubitConnect", and then create a folder called "ConfigFiles" inside that. The Folder path should be "C:\CubitConnect\ConfigFiles"
9. Copy the Client.exe, Filename.txt, Volume.txt, and Workspace.txt from the CD and place it inside of the "ConfigFiles" folder as shown below:



Computer ▸ Local Disk (C:) ▸ CubitConnect ▸ ConfigFiles

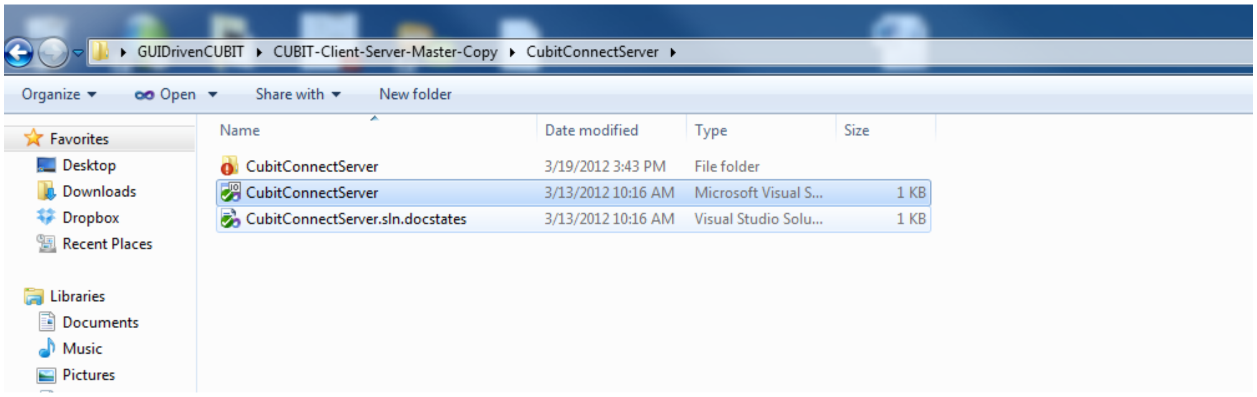
Name	Date modified	Type	Size
Client	4/7/2012 1:02 PM	Application	48 KB
Filename	4/7/2012 2:11 PM	Text Document	1 KB
Volume	4/7/2012 2:11 PM	Text Document	1 KB
Workspace	4/7/2012 2:39 PM	Text Document	1 KB

4 items

Setting up the Server

The server is a C# application and can be executed directly using the .exe file. In situations where specific ip address or port number need to be assigned, one must go through the steps below to get the application recompiled.

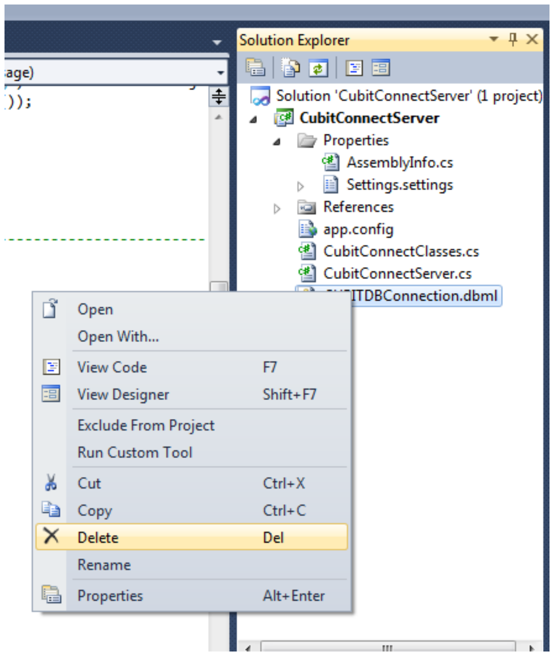
1. Open the project file, CbitConnectServer.sln, in Visual Studio 2010



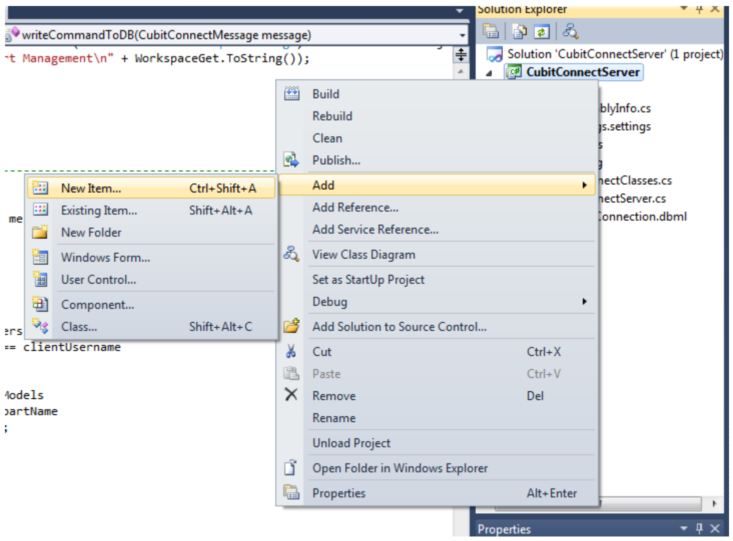
GUIDrivenCUBIT ▸ CUBIT-Client-Server-Master-Copy ▸ CubitConnectServer ▸

Name	Date modified	Type	Size
CubitConnectServer	3/19/2012 3:43 PM	File folder	
CubitConnectServer	3/13/2012 10:16 AM	Microsoft Visual S...	1 KB
CubitConnectServer.sln.docstates	3/13/2012 10:16 AM	Visual Studio Solu...	1 KB

2. Relink to the database by recreating the .dbml file
    a. Delete the current dbml file

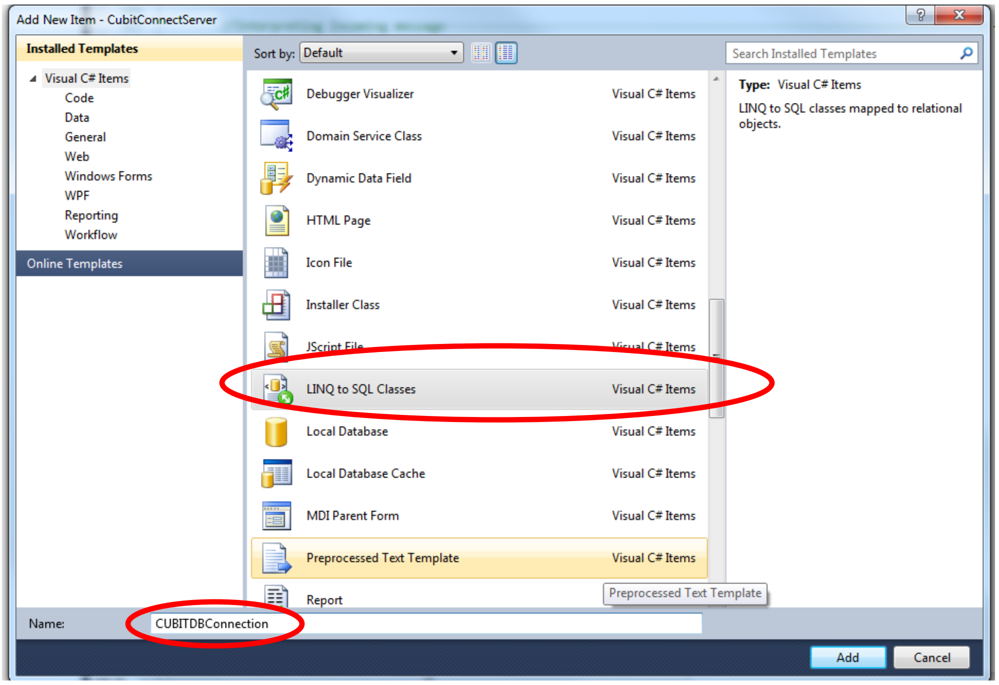


b. Right click on the project header select Add->New Item…



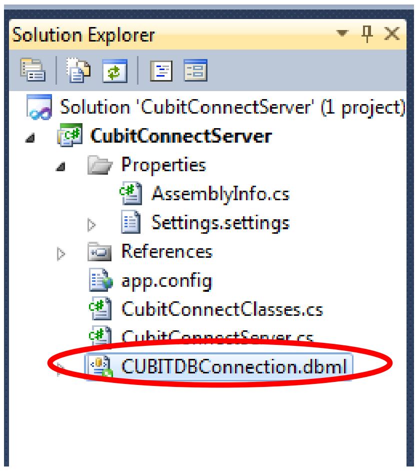
```
writeCommandToDB(CubitConnectMessage message)
t Management\n" + WorkspaceGet.ToString());
== clientUsername
```

c. Select LINQ to SQL Classes and type “CUBITDBConnection” for the Class Name.

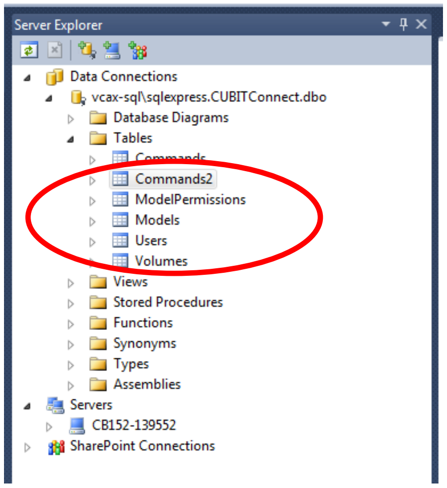

Add New Item - CubitConnectServer
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Sort by: Default
Search Installed Templates
Debugger Visualizer — Visual C# Items
Domain Service Class — Visual C# Items
Dynamic Data Field — Visual C# Items
HTML Page — Visual C# Items
Icon File — Visual C# Items
Installer Class — Visual C# Items
JScript File — Visual C# Items
LINQ to SQL Classes — Visual C# Items
Local Database — Visual C# Items
Local Database Cache — Visual C# Items
MDI Parent Form — Visual C# Items
Preprocessed Text Template — Visual C# Items
Report
Preprocessed Text Template
Type: Visual C# Items
LINQ to SQL classes mapped to relational objects.
Add
Cancel

Name: CUBITDBConnection

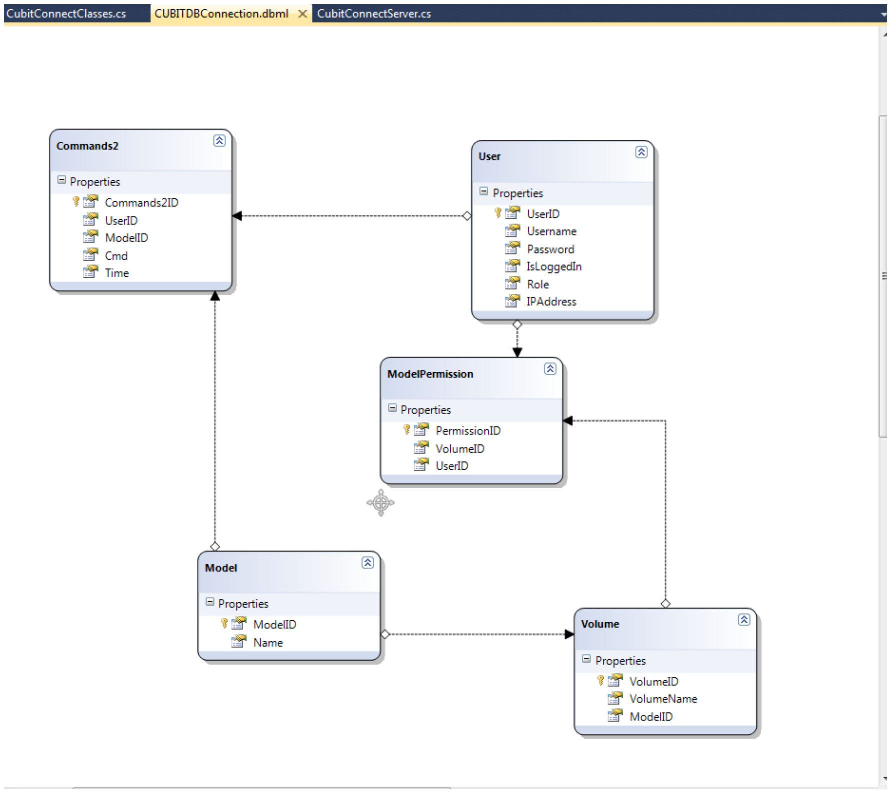
d. Double Click on the new dbml file created



e. Drag and drop Commands2, Model Permissions, Models, Users, and Volumes tables into the white space



f. The Tables should be automatically linked like the picture shown below.

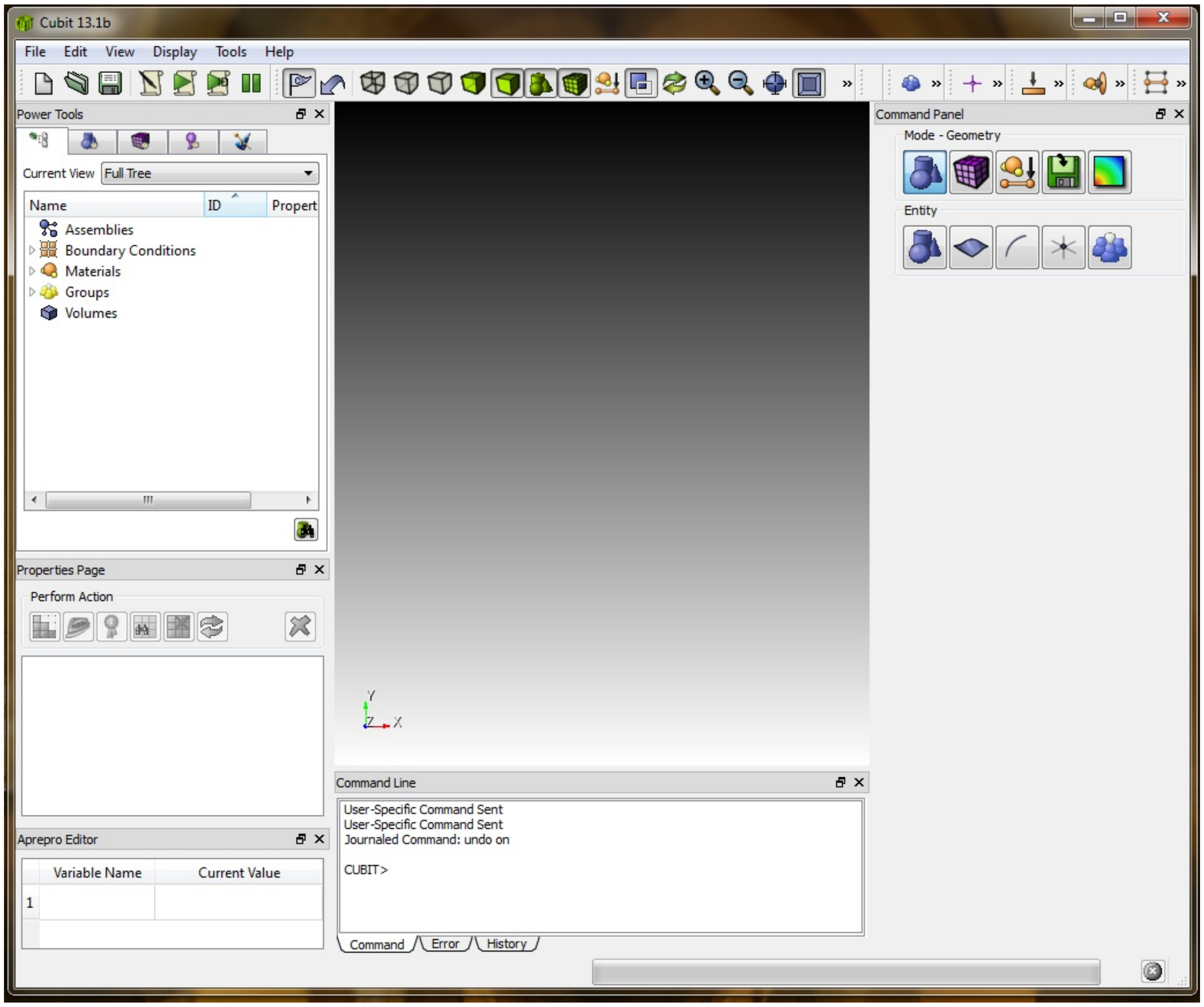


3. If the tables are linked, click “Build Solution” from Build menu.

4. If build succeeded, click ▶ to run.

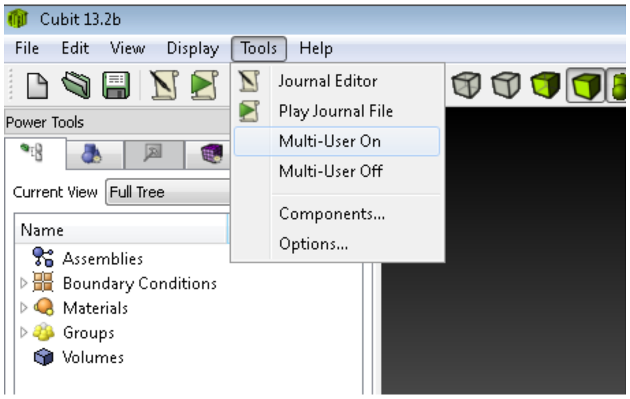
C.5 Test Tutorial

1. To run CUBIT Connect V3.0, double click on “Cubit” from -> All Program -> CUBIT Version x.x CUBIT -> Cubit, and the CUBIT GUI should look like this:

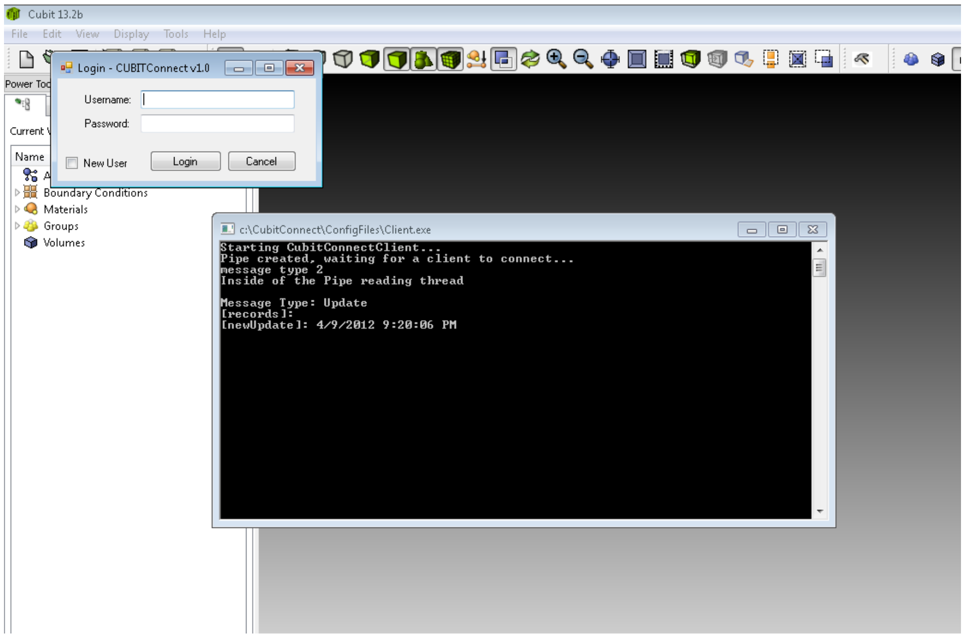


```
User-Specific Command Sent
User-Specific Command Sent
Journaled Command: undo on

CUBIT>
```

2. To start multi-user mode, go to “Tools” -> “Multi-User on”



3. A login & a console window would pop up as shown below:



```
Starting CubitConnectClient...
Pipe created, waiting for a client to connect...
message type 2
Inside of the Pipe reading thread

Message Type: Update
[records]:
[newUpdate]: 4/9/2012 9:20:06 PM
```

4. Trouble Shoot
   I. If only the console window shows up, just close the multi-user mode and try again. To turn off the multi-user mode, go to “Tools” -> “Multi-user off ”.
   II. If neither window shows up, then check the file directory where your Client.exe is located. Make sure everything is spelled correctly.

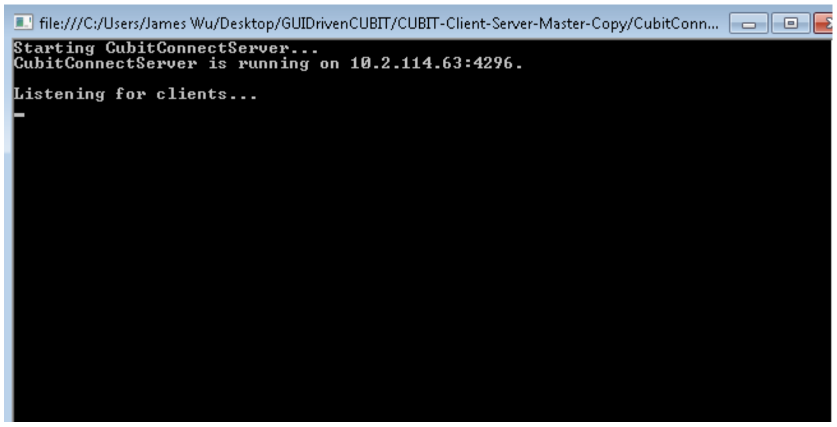
Contact Prasad or James if you have questions on setting up Cubit

For Cubit Support:

Email cubit-dev@sandia.gov or contact Dr. Karl Merkley or Mark Dewey at **Computational Simulation Software, LLC** on 801-717-2296

C.6 Demo Tutorial

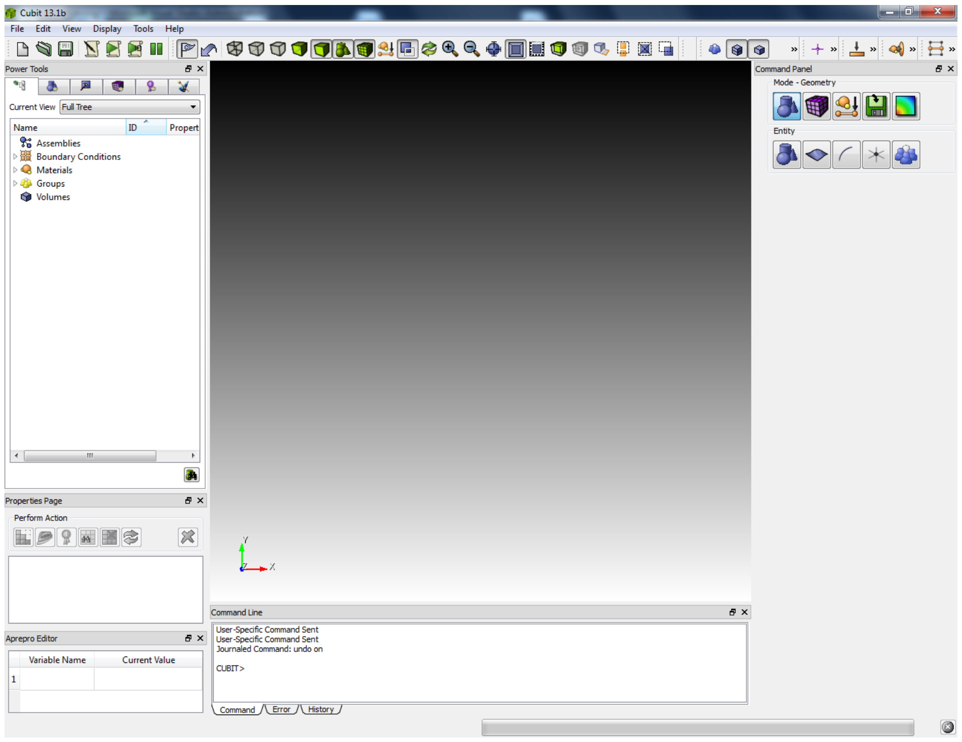
6. Start up the server, the console application will link to the database and wait for client connection.



file:///C:/Users/James Wu/Desktop/GUIDrivenCUBIT/CUBIT-Client-Server-Master-Copy/CubitConn...

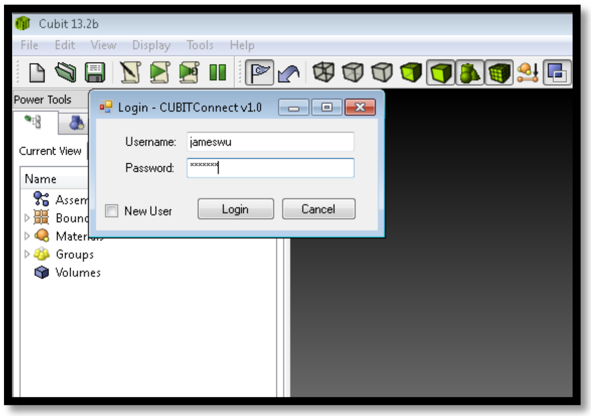
```
Starting CubitConnectServer...
CubitConnectServer is running on 10.2.114.63:4296.

Listening for clients...
_
```

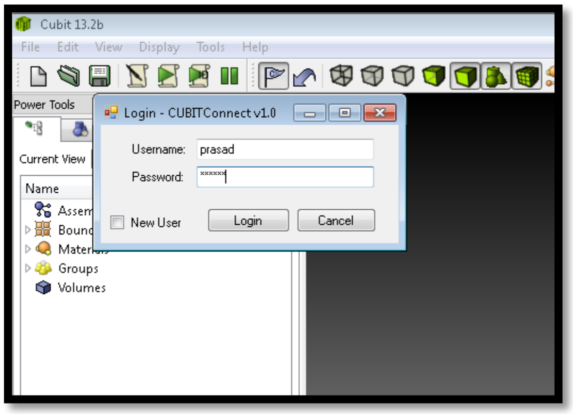
7. Start CUBIT on three different machine.

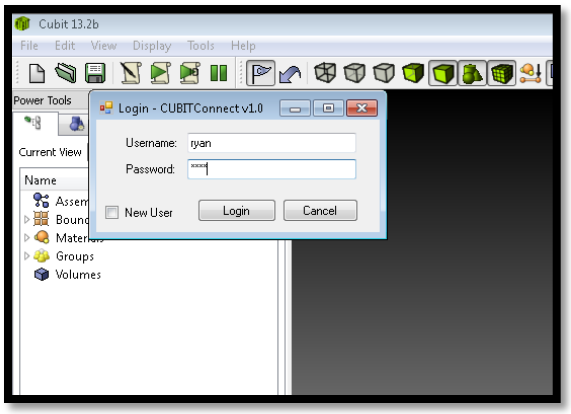


```
User-Specific Command Sent
User-Specific Command Sent
Journaled Command: undo on

CUBIT>
```

8. Each User Login into there respective account

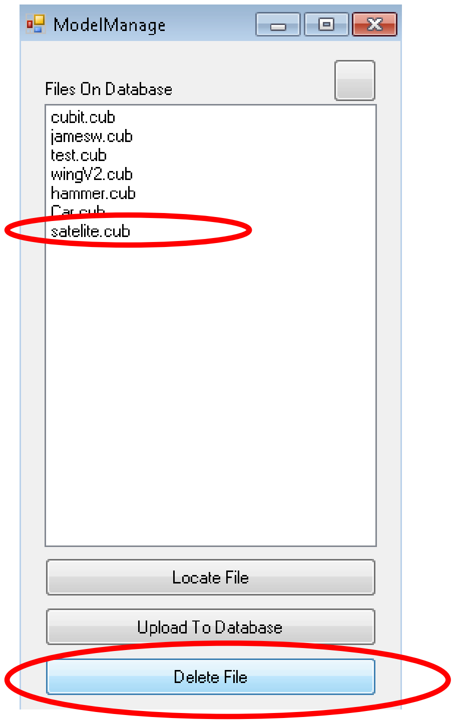






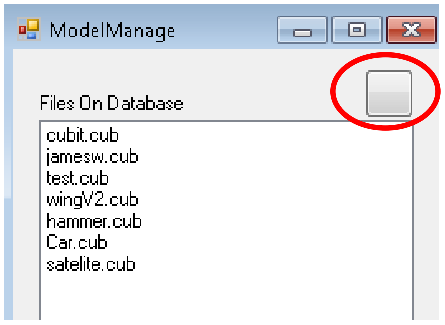
9. Instruction for Administrator

a. If “satellite.cub” is already in the file database list, One person click “Delete File”.



cubit.cub
jamesw.cub
test.cub
wingV2.cub
hammer.cub
Car.cub
satelite.cub

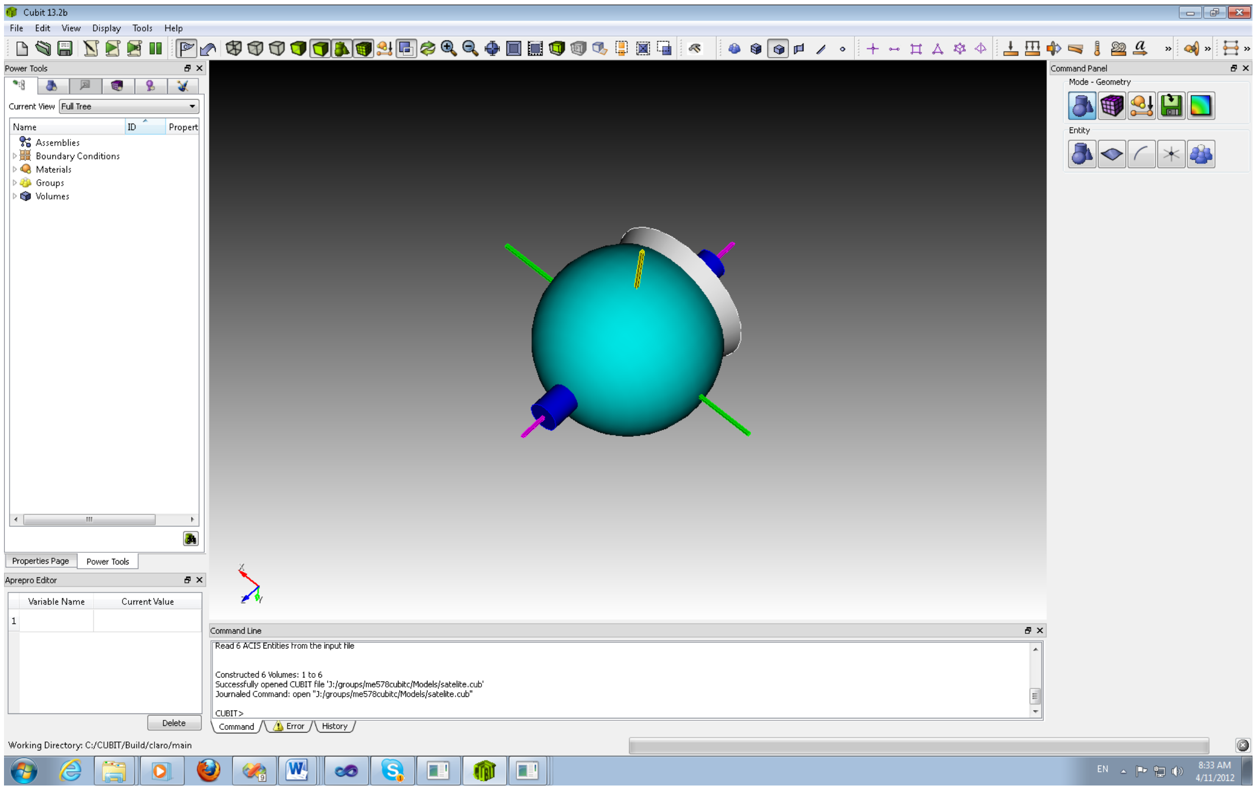
b. Click on “Refresh” if the file is still in the list



Files On Database

cubit.cub
jamesw.cub
test.cub
wingV2.cub
hammer.cub
Car.cub
satelite.cub

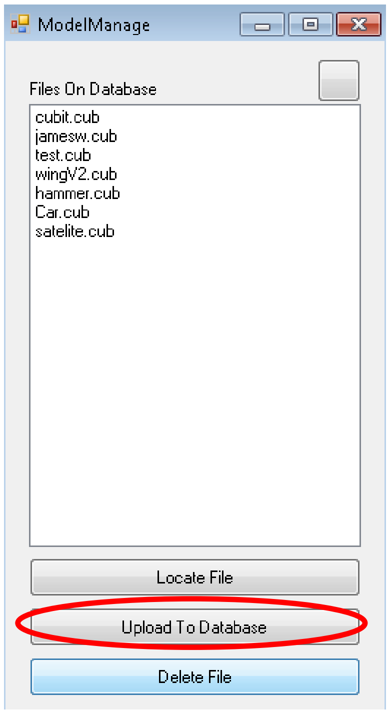
c. Open the satellite model in CUBIT use File->Open



```
Read 6 ACIS Entities from the input file


Constructed 6 Volumes: 1 to 6
Successfully opened CUBIT file 'J:/groups/me578cubitc/Models/satelite.cub'
Journaled Command: open "J:/groups/me578cubitc/Models/satelite.cub"

CUBIT>
```

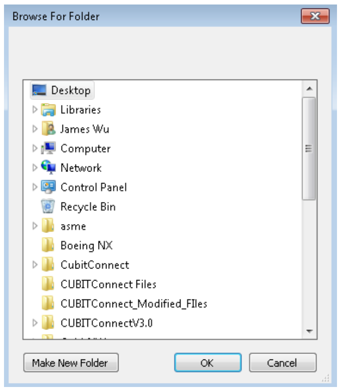
Working Directory: C:/CUBIT/Build/claro/main

d. Click “Upload To Database” to upload the model



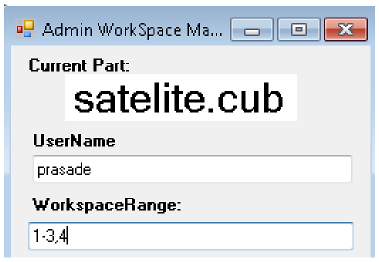
cubit.cub
jamesw.cub
test.cub
wingV2.cub
hammer.cub
Car.cub
satelite.cub

e. Locate the “satellite.cub” on your computer use “Locate File”



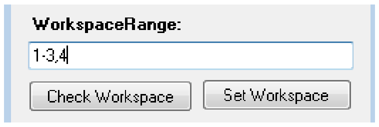
f. Double click the “satellite.cub” to open the workspace window

g. Type the username for user 1 and the type “1-3,4” for the workspace



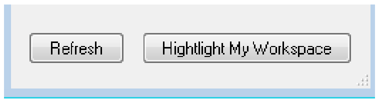
Current Part:
satelite.cub
UserName
prasade
WorkspaceRange:
1-3,4

h. Click “Check Workspace” and then hit in CUBIT to see the workspace highlighted. If you are satisfied with the workspace, then click “Set Workspace” to send the information to the database



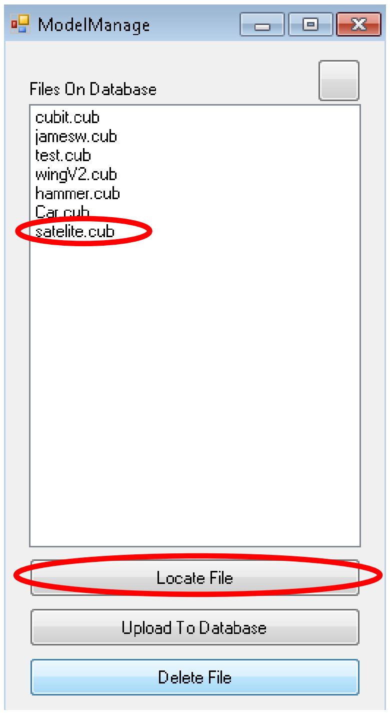
i. Change the username and workspace range and Repeate steps g. and h. for user 2. The workspace range is “5,6”

j. To highlight admin’s workspace, click “Highlight My Workspace”. For admin, the entire model should be highlighted.

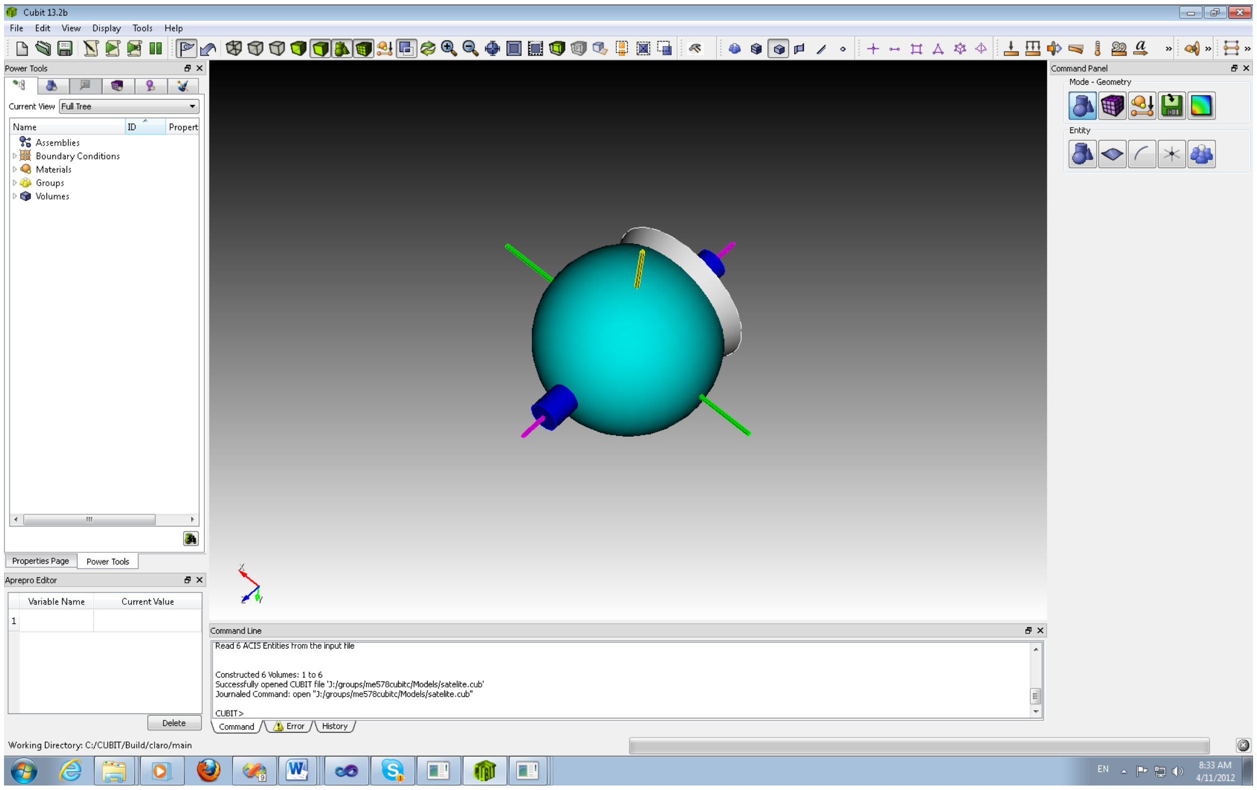


10. Instruction for user

a. After Admin finished the setup, user1 and user2 can locate “satellite.cub” on their own computer and double click the filename to open the part.

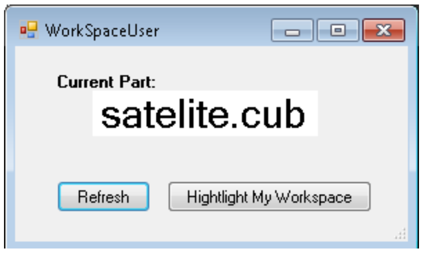


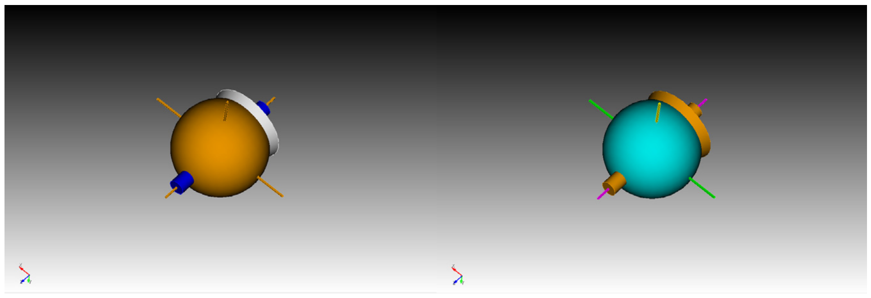
b. Click in CUBIT to load the file



```
Read 6 ACIS Entities from the input file


Constructed 6 Volumes: 1 to 6
Successfully opened CUBIT file 'J:/groups/me578cubitc/Models/satelite.cub'
Journaled Command: open "J:/groups/me578cubitc/Models/satelite.cub"

CUBIT>
```

c. To highlight the workspace pertaining to the user, click "Highlight my workspace".





d. The users are restricted to the highlighted areas. If workspaces is not highlight it, click refresh to re-sync with database

e. Mesh in the highlighted region.

C.7 Conclusions:

Multi-user CAx is a revolutionary approach to minimizing product development times and also, to creating better products. This paper has demonstrated a robust design to decompose a model by features. By restricting users to their respective workspaces at the beginning of each project design, one can eliminate many of the unintentional interference among the users. Although setting up the workspace can take up a certain amount of time, the time and confusion avoided when verbal assigning work tasks clearly outweighs extra time project managers spent putting together the decomposed multi-user environment.

C.8 Future Recommendations:

- Develop a algorithm so the workspace can be automatically and strategically generated based on geometry features
- Build the Forms inside of CUBIT using QTGUI because CUBIT main thread should be locked during the login process
- Develop another pipe connection using Qsocket, so the CUBIT function can be directly called instead of going through a static queue. (CubitGUI.cpp is an ideal location to implement this change)
- Change to Workspace Admin GUI so usernames can be a pull-down menu of all the users registered in the database
- Think about how undo commands would work