A Networking Architecture for a Multi-User FEA Pre-Processor

Prasad Weerakoon[1], James Wu[1], Tim Bright[1], Chia-Chi Teng[1], Ed Red[1], Greg Jensen[1] and Karl Merkley[2]

[1]Brigham Young University, prasadwee@yahoo.com, jw207427@gmail.com, timabright@gmail.com, ccteng@byu.edu, ered@byu.edu, cjensen@byu.edu
[2]Computational Simulation Software, LLC. karl@csimsoft.com

ABSTRACT

Current computer-aided engineering applications (CAx) only allow a single user to create/manipulate geometry and prepare the model/assembly for analysis at a given time. These engineering applications do not support multiple users to make changes to the model/assembly simultaneously. Though there are some collaborative tools embedded within these applications, they are limited to simple interactions such as screen sharing, conference calling, instant messaging, etc. In the Product Development (PD) cycle, a considerable amount of time is spent in preparing the model/assembly for Finite Element Analysis (FEA). This paper discusses the architectural considerations that were taken into account to allow multiple users to work on the same model in CUBIT (a commercial FEA mesh generation software).

Keywords: multi user architectures, ipc, named pipes, collaborative architecture, CAx.
DOI: 10.3722/cadaps.2013.527-540

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. Though 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 industrial engineering process. An example of this would be the advancements in massively multiplayer online role-playing games (MMORPG) such as World of Warcraft and Everquest 2, etc. The authors understand that it is not a simple task to convert traditional CAx tools to an environment similar to that of an MMORPG, but most of the networking/programming principles behind the MMORPG platforms can 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 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 element in FEA [11]. 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 who 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 CAD software as well [7].

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 [12]. 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 [2].

Furthermore, research has been conducted, at Beijing Jiaotong University and Texas Tech, on making FEA collaboration more effective by integrating multiple CAx tools to FEA programs at various stages of the analysis phase of the PD cycle [20]. However, none the aforementioned studies allows real-time collaborative model modification. Google Docs is a well-known example of real time multi-user collaborative environment. However, implementing a similar architecture in CAx applications is not ideal.

The Collaborative CAx methods developed by the 3-CAx (new-CAx) at Brigham Young University [13], which were primarily focused on CAD software [14], would be greatly beneficial for the field of multi-user FEA. Although FEA programs are similar in some ways to the CAD software, they have to be treated separately when studying methods of developing multi-user environments.

2 MOTIVATION

The ability to have more than one person work collaboratively on a task has proven to decrease its 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, in Finite Element modeling and simulation, most of the time was being used [11]. It was found that 73% of the time was consumed in developing the analysis solid model, meshing, 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 solid model. A multi-user pre-processing approach such as CUBIT Connect, as mentioned later in this paper, would be an ideal solution to allow multiple engineers to go in and tackle the problem of cleaning up the single solid model, thus reducing the pre-processing time significantly.

If multiple users can access and create pre-determined components of a product at the same time, and they are able to see the entire model on their screens, it would undoubtedly decrease the modeling time and enhance the entire engineering change order system. Since engineers would be able to see each otherís 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 goal behind this is to understand the research that needs to be done to develop methods of assigning workspaces, decomposing a model/assembly [3], and identifying overall conflicts with multi-user CAx implementations relating to FEA pre-processing.

3 CUBIT

3.1 Overview

CUBIT is a mesh generation tool developed primarily by Sandia National Laboratories [6] and it is the primary platform on which mesh generation and geometry preparation (FEA pre-processing) are done for FEA at Sandia and its collaborating laboratories around the United States. Sandia has allowed CUBIT to be used for research purposes by providing its source code to academic institutions. Therefore, the 3-CAx research group at Brigham Young University was able to obtain the CUBIT source code for developing multi-user CAx applications. It should be noted that CUBITõs source code is not open-source, and it is owned by Sandia National Laboratories.

3.2 Cubit Architecture

The initial version of CUBIT was only a command-line mesh generation program. In 2003 a graphical user interface (GUI), named CLARO, was added on top of the command-line version to make CUBIT more user-friendly [8]. The original command driven system and the command language were preserved in this new GUI driven CUBIT through an interface called ðCUBIT-Interfaceó. 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 (Fig. 1).

The CUBIT core was written in C++, while Python and Qt Creator were used to create the GUI. Therefore the management of data transfer between lower and high-level languages is critical. In CUBIT this is accomplished using a software tool called SWIG [15].

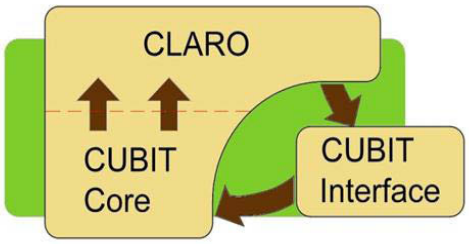


Fig. 1: How command strings are passed from GUI to Core through CUBIT-Interface.

3.3 Why Cubit?

CUBIT was used as the primary development platform for multi-user FEA pre-processing platform because of the working relationship BYU has with Sandia National Laboratories. As mentioned earlier, the 3-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. There are other multi-user CAD prototypes (NX and CATIA) also developed at BYU using 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 [14]. However, with the availability of the CUBIT source code, the multi-user version of CUBIT (CUBIT Connect) can therefore, be more robust and have more functionality.

4 CUBIT CONNECT VERSION 1.5: CLIENT SERVER IMPLEMENTATION

4.1 Early Prototype: P2P

The command strings generated by CUBIT provide an intercepting point for multi-user architectures to obtain each user's actions and transmit them to other users. As mentioned earlier, each interaction between the user and the CUBIT GUI generates a command string, and only the command string is passed down to the CUBIT core for processing. A preliminary peer-to-peer (P2P) version of CUBIT Connect was implemented to study the feasibility of a multi-user FEA pre-processor (Fig. 2). Important limitations of the P2P architecture were identified during testing.

1. No central server or manager:

All clients connect to the network and make changes to the model without any moderation in the P2P system. This is a problem if users are given access rights and allocated workspaces. Furthermore, undo sequence handling cannot be implemented on a P2P system. Adding a centralized server would add these functions, and many more, to CUBIT Connect.

2. CUBIT entity identification issues:

Even though all users had the same geometric model, the entity IDs (surface numbers, vertex numbers, etc.) were different from user to user. This will be discussed in detail later in this section.

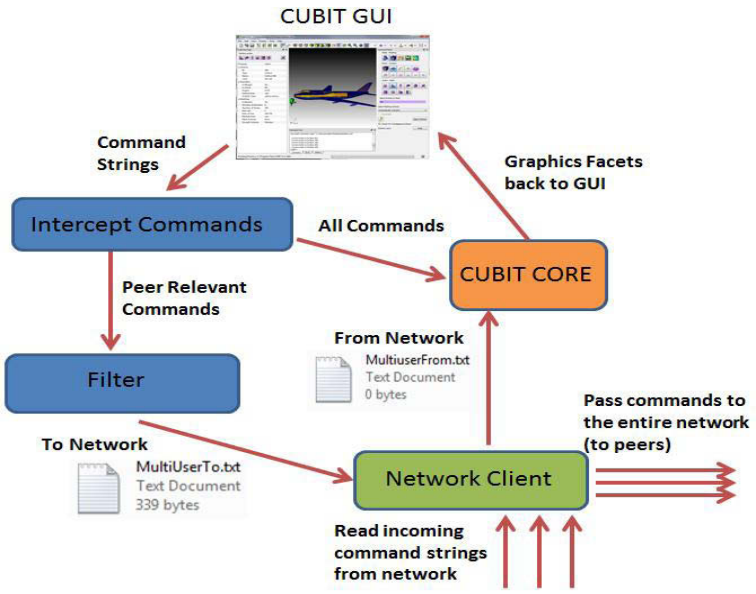


Fig. 2: Overall architecture of P2P CUBIT Connect.

4.2 Overview of the Client-Server Architecture

Although the P2P model supports the basic needs of the multi-user environment, its limitations were also apparent as illustrated in the previous section. To improve upon these limitations, a Client-Server (CS) architecture for CUBIT Connect is proposed. This architecture allows multiple clients to connect to a centralized server device dedicated to provide services. A service provided by a server

may require light computation (e.g., print servers or file servers) or intensive computation (e.g., database servers or image-processing servers). [16]

The current prototype of CUBIT Connect utilizes a CS with a thin server and strong client. The server retains all necessary function commands from each client and properly distributes them back through the networks for other clients to pick up. At the same time, the server also stores a master copy of all the commands generated into a master database for critical situations (e.g., an unexpected network complication or clients joining the process at different times). On the client side, it will obtain all of the filtered commands from the server through a first-in-first-out (FIFO) queue where data is then stored and held for processing later. Once an update notification is executed by a particular client, that client will reach into the queue and retrieve all commands from other clients for processing.

4.3 TCP/IP Implementation

Several different implementations of the CS architecture are developed to optimize the communication process, and the first implementation utilizes .NET socket namespace, which is a managed implementation of the windows sockets (Winsock) [1]. Managed means code written in high-level programming languages (C#, J#, C++ .NET, etc.) sharing a unified set of classes within the windows .NET framework [18]. By compiling CUBIT Connect in this managed environment many typical programming mistakes, which lead to security holes and unstable applications, can easily be avoided. Managed code also automatically takes care of unproductive programming codes (e.g., garbage collection, type safety checks, etc.). However, none of these capabilities are available with unmanaged code [17]. In this design, the client will initiate a connection to the IP address and the port number designated by the server. Once the server accepts this request, the hand shake process is completed and a stream is created to allow data transfer between the client and server.

Different types of messages have been established for CUBIT Connect to ensure more functionality, and the details about these message types will be discussed further in later sections. Because of these different message types, a filter and sorting system has been implemented to distinguish between them. This is accomplished by identifying clients using their distinct IP addresses. The TCP/IP server has the ability to keep track of each incoming client by their IP addresses and stores them into an array list for future reference. This means the filter resides on the server side enabling the server to decide when to store messages, when to send out updates and which client will get the updates. This implementation was successfully demonstrated during a National Science Foundation (NSF) sponsored conference held at BYU during the month of October 2011 [10]. During this event, several industrial and university representatives were able to have a firsthand experience using this design of CUBIT Connect.

4.3.1 Shortcomings of TCP/IP implementation

Although the TCP/IP version of the CS model provides a reliable connection between the clients and the server, it has proven to be a difficult task to embed the client program into the CUBIT source code. This is mainly due to the difference in programming languages. The TCP/IP version of CUBIT Connect is written in C# utilizing the .NET framework, which is a type of managed code, whereas CUBIT source code is written in native C++ which is unmanaged. Multiple solutions were investigated (e.g., creating dynamic-link library (DLL) files and common language runtime (CLR) projects), but integrating C# into C++ was proven to be difficult to accomplish.

Another limitation of this prototype is its platform dependency. CUBIT is a platform independent FEA pre-processor, which gives it an edge over its competitors. However, the C# CS model cannot run on other platforms (MAC OS X, Linux, etc.). To keep the cross-platform feature of CUBIT and also allow

CUBIT Connect functions to be integrated into the CUBIT source code, a named pipe version of CS model is proposed.

4.4 Named Pipe Implementation

The second implementation of the CS architecture utilizes Microsoft Windows named pipes. It is widely believed that a Client-Server system based on named-pipes is much easier to implement than any other designing methods [16]. The named pipe CS architecture is written in native C++, and thus can be easily added as an add-on to existing CUBIT. Like the TCP/IP model, the current named pipe design also involves a host computer (known as a server) dedicated to handle all of the processes, as well as several user computers (known as clients) connecting to the server on the network. Due to the inability of named pipe servers to distinguish between clients, the sorting and filtering process is built on the client side. Whenever a message is sent by the server, it is placed into the pipe for all the clients to pick up. A diagram of the named pipe design is shown in Fig. 3, and more implementation details will be explained below.

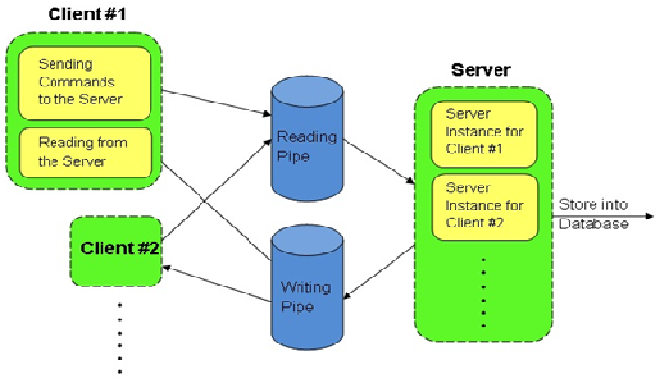


Fig. 3: Named Pipe architecture of CUBIT Connect.

4.4.1 Message types and message serialization

Before setting up the client and server individually, common message structures need to be established for both the client and the server to recognize. To accomplish this, a serialization and deserialization function is created to append messages with a message type and a unique client ID. For this CUBIT Connect prototype, there are three message types involved (1.Regular command message, 2. Master trigger, 3. Master update). Both the **ò**regular command message**ó** and the **ò**master trigger**ó** are generated on the client side; Regular command messages are messages generated from the CUBIT GUI for the CUBIT core to process (e.g., create sphere or mesh volume 1), and master trigger messages are messages initiated by the user to re-synchronize their model with the master database (e.g., In situations such as a client computer encountering unexpected difficulties or clients joining the work environment at different times). A **ò**master update**ó** is the only message type generated on the server side and can only be created when a **ò**master trigger**ó** is received. It contains all of the command messages generated during the collaboration process to ensure up-to-date information for all clients during situations mentioned previously.

If a regular command or a master trigger is picked up from CUBIT, the client will serialize the command with message type and client ID. On the server side, all incoming messages are modified by adding the serverõs current time-stamp at the end. Having the server assign system times provides

consistency across the network, and the slight variation between system clocks on each client computers can be avoided.

After the system time is attached, the server is programmed to store a copy of the message string into the master database and then place the processed string back into the pipe for other clients to pick up. Occasionally the incoming message is a master trigger instead of a regular command message. In that case, the server acts accordingly by making a master update as mentioned before. The serialization process for master copies is slightly different than the previous two message types (Fig. 4):

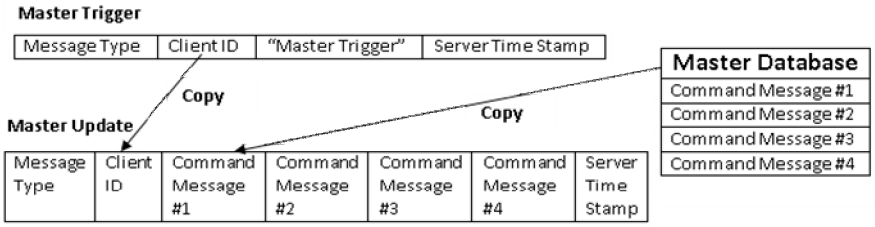


Fig. 4: Master Update Message Structure.

1. A different delimiter is used to separate each component of the master copy.
2. The server copies the unique ID on the master trigger and places it into the master update as its unique ID.
3. All of the messages saved on the database are placed into the **ò**master update**ó** separated by the delimiters. (Note: The master update massages can be significantly long. Therefore, it is recommended to not set a buffer size to avoid messages getting truncated).

Once the serialization processes are setup, the deserialization processes are simple tokenize functions which separate message components by the specific delimiters. The serialization and de-serialization functions not only increase the intelligence of this CS model but also leave room for future improvements (e.g. instant messages between users, VOIP packets can be another message type implemented in the future).

4.4.2 Named pipe server

This model of named pipe server is designed to have minimal capability. Its basic functionalities are generating named pipes, storing messages to the database, and sending commands back into the pipe. Whenever the server starts up, it immediately generates instances of two named pipes: one called reading pipe and another called writing pipe. Once a client initiates a connection, a Client Thread (CT) is generated, which is separate from the main thread. A detail implementation of the thread is discussed in later sections. While the CT separates and interacts with the client on its own, the main thread loops and creates another instance of both reading and writing pipes waiting for new clients to connect. For this design, there are no thread limits or pipe instance limits assigned, which means the server is capable of handling as many users as the network allows.

4.4.3 Named pipe client

The named pipe client consists of two running threads, one dedicated for sending commands and another dedicated for receiving. Whenever a client connects to both reading and writing pipes generated by the server, it will create a Client Listening Thread (CLT) dedicated to check the pipe for incoming messages from the server. Again, the detailed implementation of the thread will be discussed later on. While the CLT is generated, a unique ID and time-stamp variable are also created to

distinguish between users. After the ID and time-stamp variable are stored, the client is ready to communicate with the server. As mentioned earlier, either a regular command message or a master trigger will be sent from the client to the server.

While the main thread writes to the server, the CLT is constantly reading for incoming messages in the writing pipe. As explained earlier, one of the limitations of named pipe servers is its inability to distinguish between clients, thus incoming messages will require the client themselves to filter and sort. This constant reading process is built into a while loop, and the resulting command messages will be added to a Client Queue (CQ). Once an update is initiated by the user, the client will go through the CQ and update accordingly. Details about the update feature will be discussed later on.

4.4.4 Threading

As mentioned previously, the integration of multiple threads is the key to the efficiency of this CS model. Like many current software applications, CUBIT Connect is also takes advantage of multiple cores on modern computers which enables parallel execution and reduce processing time. Because of the true concurrent nature of the thread processes, it is important to layout the architecture carefully beforehand to avoid race conditions and thread safety issues (e.g., some sequential functions might be out of order because some functions execute faster than the others, or common data being accessed simultaneously).

The thread function on the client side is dedicated to read messages coming from the server, thus the pipe handle for the writing pipe needs to be passed into the thread as a parameter. On the server side, threads are created to handle each instance of pipe assigned to each client. When a client initiates the connection to the pipe instance, a CT is generated to handle that particular instance. Unlike multi-threading on the client side, the threads on the server side require both reading and writing pipe handles to be passed for reasons mentioned previously. This brings up a complication since thread functions only allow one parameter to be passed into the function [5]. To solve this issue, a structure such as a vector, list, etc., must be created to group both handles into one parameter.

Another complication of having multiple threads is thread safety. Normal compilers will force the thread function to be static which limits its ability to access non-static members in the same class and vice versa. This brings up a dilemma because the data read from the pipe cannot be stored in a way which is accessible for other none-static functions to retrieve. There are several different ways to get around this problem. But for this CUBIT Connect model, a singleton class is incorporated. A singleton is a class design which restricts the instance of a class to one object, and it can act as an intermediate member between the static and non-static members inside of the server to ensure that common data is accessible by both static and non-static members [9].

It should be noted that, there are many different ways to implement threading into CUBIT Connect, and the current thread implementation may not be the most effective. Therefore, multiple efforts are still underway to explore additional designs to improve the capability and efficiency of the processing thread, so a faster and more efficient multi-user environment can be achieved.

4.5 Manual Update

The original intent was to have an automatic update function for the CUBIT Connect prototype so whenever a user makes a change to the model, it updates other users' computers instantly. However, some complex meshing algorithms can take an enormous amount of time to execute. This is especially the case when the model/assembly is large and complex. The problem is further escalated because the CUBIT core and CUBIT GUI run on the same thread, restricting the user to work on their model until the calculation is completed. Because of this limitation, the current CUBIT Connect prototype uses a manual update feature to import commands from other users. When the user types

the command ‘update’ on CUBIT’s command line, all the commands by the other users will be executed by the core sequentially.

Although the current version of CUBIT has limitations with both the GUI and core running on the same thread, effort is underway to make CUBIT run on multiple threads. If this happens in the near future, then some kind of automatic update function could be implemented to update in real-time. But again, the complex nature of meshing algorithms can still tie up a considerable amount of system resources, which might cause the client computer to freeze while an algorithm is running in the background. Therefore, even if the core and the GUI ran on separate threads, the manual update function is still recommended.

4.6 Named Pipe Working Prototype

The named pipe CUBIT Connect was also tested allowing multiple users to create geometry, mesh, and setup the model/assembly for analysis. As mentioned, the key to the entire process is to define the user’s work space beforehand to avoid conflicts during multi-user collaboration. Fig. 5 shows three work spaces defined on CUBIT Connect for three different users operating on the same race car model.

4.7 Limitations

From our experimentations, the named pipe version of CUBIT Connect does solve the limitations presented by the previous versions. For example: the named pipe design utilizes CS architecture which centralizes data processing for simpler and efficient data management, it is written in native C++ which can be easily implemented into CUBIT source code, and it can be easily switched out with sockets to operate on other platforms. Despite these improvements, there are other issues need to be addressed in future prototype designs.

4.7.1 Manual update

Both P2P and the CS versions of CUBIT allows only a manual update option where each user has to type a command on CUBIT’s console to pull all the updates from other users to their computers and update the model. The reason being each computer has to process all the commands locally and that process may take time depending on the command type. Some of these meshing commands may take hours or days to complete depending on the complexity of the operation. Therefore, there is no automatic update option to see the model changes in real time.

4.7.1 CUBIT entity identification system unpredictability

Each CAx system has some sort of entity identification to identify difference volumes, faces, nodes, etc., created by the user or as a result of a computation. This is true for CUBIT as CUBIT assigns an integer ID to the geometry, meshing, and boundary features (Fig. 6).

Whenever an additional command is executed upon these geometries, they will identify the feature by the integer ID The problem arises because integer IDs are sequentially assigned by the order of feature execution. When one client updates the commands from other users, the CUBIT core will reassign feature IDs based on their order of execution which results in different feature IDs for different client. This complication can be easily fixed by adding a unique client identification to the feature IDs generated in CUBIT core. This method would keep the feature identification uniform across the network. Another solution to the problem involves a redesign of client/server architecture, which will be discussed in the future implementation section.

4.7.2 Redundant

As mentioned earlier, each client computer has to compute all the command strings from each computer. This is found to be redundant as valuable time is lost while these computations are

executed. These computations also freeze the model while they execute, disabling the user's GUI. This is a limitation of CUBIT as the program is not multi-threaded (i.e. the graphics process and the computation process run on the same thread and one has to finish executing for the other to be used).

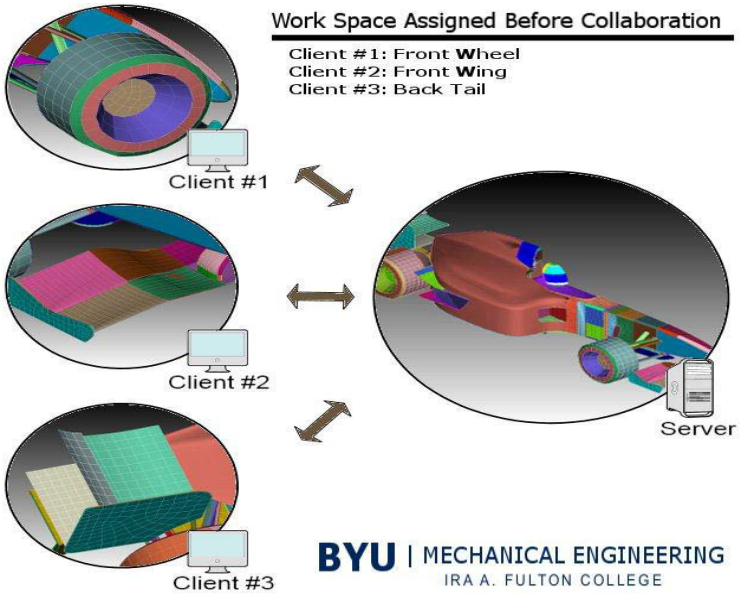
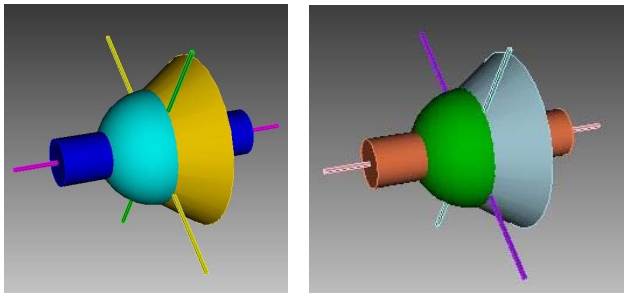


Fig. 5: CUBIT Connect pre-determined workspaces.



(a) (b)

Fig. 6 a, b: CUBIT Connect ID issue example: Two users in a multi user CUBIT session create multiple volumes that lead to the overall model as shown in (a) & (b). Notice the changes in color of the volumes on each user's screen. CUBIT has a color scheme to denote different volume numbers and the change in colors of the volumes on each user's screen denotes difference in volume numbers.

4.7.3 Workspace assignment/control

The current CUBIT Connect prototypes also lack features for defining work spaces. During experimentation, work spaces are identified for users by defining a general work area which are distant and not overlapping so that interference can be avoided. In future implementations, features should be developed to enable strategic definition of workspaces which will constrain the user to a

particular work area. Any modification to the model outside of the assigned workspace should be prohibited.

4.7.4 Undo function

The last major limitation for the current CUBIT Connect prototype is the undo feature. Undo sequences can be complicated in a multi-user environment. Since the last command executed may not be the last command generated by the client, it might disrupt other users from doing their work. Researchers have proposed an undo command handle method for the next implementation of CUBIT Connect and are working on the final details.

5 FUTURE WORK: CUBIT CONNECT (CLOUD COMPUTING)

As mentioned in the previous section, the next implementation of CUBIT Connect is currently under development, and it will target towards minimizing and solving issues such as entity identification, work space allocation, and undo sequence handling.

5.1 Overview

The next prototype consists of a strong server and thin client, where the CUBIT GUI resides on the client side and the CUBIT core resides on the server side. That means the server will handle all computations. This implementation requires CUBIT to be cleanly separated between the GUI and core so both components can execute separately on separate machines (Fig. 7). A diagram of the new architecture is displayed in Fig. 8.

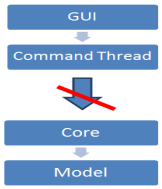


Fig. 7: Severing the connection between CUBIT Core and GUI.

5.2 Cloud Serving Architecture

The server bridge shown in Fig. 8 will have the capability to manage network traffic: It can order the incoming messages into a command queue, it can notify an administrator when traffic is approaching the processing limit, and it can equally distribute the load among CUBIT cores to minimize processing time. The intension for this design 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 to the server to reduce the work load. This will allow the resources to be dynamically allocated, so the minimum amount of resource is used to perform the task.

5.3 Improvements

5.3.1 Entity Identification system

Since there is only one master model which all clients are synchronized to, the ID system should stay uniform across clients.

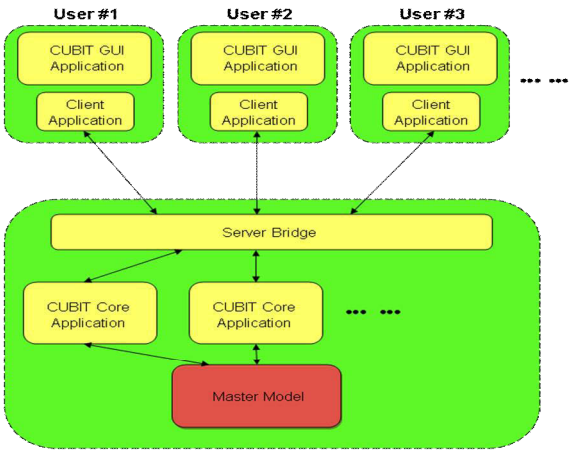


Fig. 8: Cloud Computing CUBIT Connect architecture.

5.3.2 Thin clients

Since just the GUI portion is residing on the client computer, expensive computer hardware is not necessary. This means reduced cost of maintenance on client computers and the owners can invest more money on acquiring and maintaining a central server/super-computer.

5.3.3 Scalability

As mentioned earlier, the server would be able to dynamically allocate computation cores according to demand. This means in its full, robust implementation stage, hundreds of engineers would be able to work on the same model/assembly in a multi-user setting.

5.3.4 No redundant

One master model means there are no redundant calculations as opposed to the current implementation. Each user's modifications are computed by the parallel CUBUIT cores residing on the server, and they will constantly update the master model so each command will only execute once. There arises an important problem when multiple CUBIT core instances are modifying an updating a command master model file. That is, how to handle conflicting model changes before they actually change the model. Using instant messaging or a VOIP service such as SKYPE to communicate between users while they make changes to the model have been discussed [19]. Real-time screen sharing allowing users to see others' perspective can also, lead to better conflict resolution.

5.3.5 Manual Update

In this future implementation, only the updated graphics facets are passed from the server to the client. This means there are no large-scale computations done on the client side. This should solve the problem of complex algorithms freezing up computer resources, thus the clients should be able to see modifications to the overall model/assembly in real-time. Therefore, the manual update command will no longer be needed. A user would be able to work on their assigned workspace while others' workspaces are constantly getting updated. This would allow users to spot potential problems created by adjacent users which allow them to quickly and effectively convey those concerns to their co-workers.

5.3.6 Cloud Computing

A highly talked about topic in today is cloud computing, and this version of CUBIT Connect could easily be implemented in a cloud environment. This will inherit all the benefits of cloud computing such as the elimination of owner operated servers, better maintenance, no down-time, etc. [4].

6 CONCLUSIONS

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

This paper has covered several network architectures and prototypes to measure the usability of CUBIT and uncover any potential research areas that relate to multi-user collaboration. Even though the current version of CUBIT Connect is not robust enough for industry implementation, the idea is to motivate commercial CAx software developers to start developing similar collaborative environments with the assistance from this research group at BYU.

REFERENCES

[1] About Winsock, http://msdn.microsoft.com/en-us/library/windows/desktop/ms737523%28v=vs.85%29.aspx , Microsoft Developer Network (MSDN).

[2] Wu, A.; Geng, Y.; Zhang, G.; Wang, J.: An Agent-Based Collaborative FEA System for the Optimal Design of Electromagnets, International Conference on Power System Technology, 4, 2002, 2150-2154. DOI: 10.1109/ICPST.2002.1047162

[3] Bu, J.; Jiang, B.; Chen, C.: Maintaining Semantic Consistency in Real-Time Collaborative Graphics Editing Systems, International Journal of Computer Science and Network Security (IJCSNS), 6(4), 2006, 57-61

[4] Chappell, D.: Windows Azure and ISVS: A Guide for Decision Makers, Microsoft, 2009

[5] Creating Threads, http://msdn.microsoft.com/en-us/library/windows/desktop/ms682516%28v=vs.85%29.aspx, Microsoft Developer Network (MSDN).

[6] CUBIT, http://cubit.sandia.gov/, Sandia Corporation.

[7] Lee, H.; Kim, J.; Banerjee, A.: Collaborative Intelligent CAD Framework Incorporating Design History Tracking Algorithm, Computer-Aided Design, 42, 2010, 1125-1142. DOI: 10.1016/j.cad.2010.0.001

[8] Merkley, K.-G.: The Tool Set for Building Claro ð A Component Loading Architecture, Sandia National Laboratories, Albuquerque, NM, 2006, http://cubit.sandia.gov/documents/ClaroWhitePaper.pdf

[9] Meyers, S.; Alexandrescu, A.: C++ and the Perils of Double-Checked Locking, Dr. Dobbðs Journal, July & August 2004, http://drdobbs.com/cpp/184405726

[10] Multi-user Testbed Objectives, http://www.et.byu.edu/~ered/e-DesignOctIAB/html/prototypes.html, Center for e-Design.

[11] Owen, S.-J.; Clark, B.-W.; Melander, D.-J.; Brewer, M.; Shepherd, J.-F.; Merkley, K.; Ernst, C.; Morris, R.: An Immersive Topology Environment for Meshing, 16th International Meshing Roundtable, Seattle, WA, 2008, DOI: 10.1007/978-3-540-75103-8_31

[12] Peng, J.; McKenna, F.; Fenves, G,-L.; Law, K,-H.: An Open Collaborative Model for Development of Finite Element Program, Proceedings of the Eighth International Conference on Computing in Civil and Building Engineering, 2000, 1309-1316. DOI: 10.1061/40513(279)171

[13] Red, E.; Holyoak, V.; Jensen, G.; Marshall, F., Ryskamp J., Xu Y.: A Research Agenda for Collaborative Computer-Aided Applications, Computer-Aided Design and Applications, 7(3), 2010, 387-404. DOI: 10.3722/cadaps.2010.387-404

[14] Red, E.; Jensen, G.; French, D.; Weerakoon, P.: Multi-user Architectures for Computer-Aided Engineering Collaboration, 17th International Conference on Concurrent Enterprising (ICE), Aachen, Germany, 2011

[15] Simplified Wrapper and Interface Generator, http://www.swig.org/, SWIG.

[16] Sinha, A.: Client-Server Computing, Communications of the ACM, 35(7), 1992, 77-98. DOI:10.1145/129902.129908

[17] System.Net.Sockets Namespace, http://msdn.microsoft.com/en-us/library/sb27wehh.aspx , Microsoft Developer Network (MSDN).

[18] What is Managed Code?, http://msdn.microsoft.com/en-us/library/windows/desktop/bb318664%28v=vs.85%29.aspx, Microsoft Developer Network (MSDN).

[19] Xu, Y.; Red, E.; Jensen, C.-G.: A Flexible Context Architecture for a Multi-User GUI, Computer-Aided Design & Applications, 8(4), 2011, 479-497. DOI: 10.3722/cadaps.2011.479-497

[20] Yu, J.; Cha, J.; Lu, Y.; Xu, W.; Sobolewski, M.: A CAE-integrated distributed collaborative design system for finite element analysis of complex product based on SOOA, Advances in Engineering Software, 41(4), 2010, 590-603DOI: 10.1016/j.advengsoft.2009.11.006