Brigham Young University
BYU ScholarsArchive

All Theses and Dissertations

2012-02-28

Integration of Massive Multiplayer Online Role-Playing Games Client-Server Architectures with Collaborative Multi-User Engineering CAx Tools

Joshua D. Winn
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Mechanical Engineering Commons

BYU ScholarsArchive Citation

Winn, Joshua D., "Integration of Massive Multiplayer Online Role-Playing Games Client-Server Architectures with Collaborative Multi-User Engineering CAx Tools" (2012). *All Theses and Dissertations*. 6209.
https://scholarsarchive.byu.edu/etd/6209

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Integration of Massive Multiplayer Online Role-Playing Games Client-Server Architectures with Collaborative Multi-User Engineering CAx Tools

Joshua D. Winn

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

Master of Science

C. Greg Jensen, Chair
W. Edward Red
Chia-Chi Teng

Department of Mechanical Engineering

Brigham Young University

April 2012

Copyright © 2012 Joshua Winn

All Rights Reserved

ABSTRACT

Integrating Massive Multiplayer Online Role-Playing Games
Client-Server Architecture with Collaborative Multi-User
Engineering CAx Tools

Joshua Winn
Department of Mechanical Engineering, BYU
Master of Science

This research presents a new method for integrating client server architectures that are used for the development of Massive Online Role Playing Games (MMORPG) into multi-user engineering software tools. The new method creates a new architecture named CAx Connect by changing the client-pull-server communication pipeline to a server-push-client communication pipeline, effectively reducing the amount of bandwidth consumed and allowing these tools to utilize multiple server processors for complex calculations. This method was used on the new NX Connect multi-user CAx prototype developed at BYU.

The new method provides a road map to further implement this architecture and its services into additional multi-user CAx tools. To demonstrate the effectiveness of this technology, a prototype architecture was built to provide a front end service, a message relay service, and a database insertion service, which were integrated into the current architecture. The front end service provides load balancing of clients, while the feature administration service passes messages throughout the architecture. The database insertion service inserts features passed from the NX Connect client into the database. The results show that this architecture is more efficient and that a scalable architecture was created, successfully demonstrating the integration of this architecture with multi-user CAx tools.

Keywords: multi-user CAD, collaboration, MMORPG

ACKNOWLEDGEMENTS

I would like to thank my wife Katie for her support of my pursuit of higher education and for her encouragement. I wouldn’t have gotten here without her. I would like to thank my extended family for their encouragement. I would also like to thank my advisor Dr. Jensen, Dr. Red, and especially Dr. Teng for his guidance, Tim Bright, and the people at the Ryzom Community for their help and support. Lastly, I would like to thank those that supported this research through funding, including the National Science Foundation and members of the BYU I/UCRC: Belcan, Pratt and Whitney, Boeing, Microsoft, PCC Airfoil, and Spirit Aerosystems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Collaborative engineering tools are in high demand by global corporations as they attempt to reduce time to market and optimize the product design process. As a result, collaborative tools are evolving to meet this demand. However, current collaborative tools require a sustainable architecture to support their advanced capabilities. Current architectures have been demonstrated for asynchronous work, or collaborative work that is not done at the same time. However, these architectures are insufficient for upcoming new synchronous collaborative tools, where collaborative work is done at concurrent times. It is necessary to utilize synchronous architectures where collaborative design is supported to meet this demand. One such architecture where collaboration has been proven is a massive multi-player online role playing game (MMORPG) client server architecture, which has proven itself for collaborative online gaming. A MMORPG architecture is a multi-server, multi-client architecture created to handle large loads and provide scalability, which are reasons why the gaming industry uses them for massive online game play. This architecture provides many advantages over other architectures and can be utilized for collaborative engineering tools.

1.1 Problem Statement

With the advent of more powerful computers, graphic cards, and communication technologies, a shift in engineering collaborative design methodology has begun. The literature states that CAx tools need support for real time collaboration (Zheng, 2008), allowing product design to occur across different languages and geography, as this is becoming increasingly common (Cera, et al. 2010). These tools all require a framework to support the many users who will be asynchronously and synchronously using the tools. The frameworks to choose from most commonly use web based systems, peer to peer frameworks, and client server architectures. Solutions have been proposed on how to utilize these frameworks for collaborative, multi-user tools. However, “a significant problem for [client server and peer to peer] architectures is that the communication efficiencies are still quite far from satisfactory when large-size feature- and assembly-based models are designed collaboratively.” (Fuh, 2004) Two obstacles to overcome are huge bandwidth consumption across communication pathways and lack of scalability.

1.2 **Research Objectives**

Fully distributed client server architectures or clusters like those found in Massive Multi-player Online Role Playing Games (MMORPGs) have proven successful for handling loads that multi-user applications create (CCP Games, 2010). Since there are multiple servers handling different aspects of the virtual world held in “the cloud” (a virtual 3D world used for gaming), there is little, if any, synchronization issues and users experience continuous gameplay. This architecture has been used to model real life scenarios (Orland, 2008) as well as provide entertainment through multi-user computer games, but has never been used for collaborative engineering purposes.

The objective of this research is to integrate MMORPG architectures with multi-user CAx tools, effectively eliminating the bandwidth bottleneck and providing scalability for interprocesses. The key to achieving this objective is to identify the necessary requirements and processes to sustain a MMORPG architecture. Our research will remove the game component and, in its place, create communication pathways for multi-user engineering software applications. Specifically the following tasks will be used to reach the thesis objectives:

1. Perform an in-depth study of fully distributed client server architectures with the intent to change the gaming environment into a fully collaborative design environment.
2. Layout network pathways to specify communication paths between servers and clients.
3. Describe necessary protocols and requirements needed to secure a fully distributed client server architecture.
4. Utilize an open source game server and create a collaborative client application in the cloud.
5. Analyze the bandwidth consumption of this environment and compare to NX Connect bandwidth consumption. This can be done using a ping command which will test latency (how long it takes a packet of information to get from a client to a server).

1.3 **Problem Delimitation**

CAx Connect, the architecture created for this research, is a prototype of a fully fledged MMORPG client-server architecture but does not have all of the services necessary for a multi-user massive online game. To create an MMORPG, some services are required to create a basic architecture. These services are discussed in the next section. This method is also only tested on Siemens UG NX 6.0 CAD running on Microsoft Windows 7 operating system which utilized NX Connect, a multi-user collaborative CAD tool developed at Brigham Young University which in its current state is not a fully robust multi-user CAD system. While this prototype is limited in scope, it should be noted that these limitations are only based on the proposal of this research, and that the technology is available to create a complete MMORPG architecture for these tools.

1.3.1 Architecture Rationalization

The MMORPG architecture was selected because of the advantages that it has over client-server and peer-to-peer networks, specifically load balancing, scalability and push technology, all of which are explained in Chapter 2. However, development of an entire MMORPG architecture would be unrealistic for this research, as the gaming component is unnecessary; thus, the related functionality will not be utilized for the integration of multi-user CAx tools. However, to reach the objective of integration with a multi-user CAx tool, some services, or functionality was implemented. The set of services, described in Chapter 3, was created to provide a point of entry for the multi-user tool, including a message passing system for multiple servers, and a database insertion system, thus allowing the MMORP server to have a skeleton framework to begin development. Features support was also created, so that the multi-user tools can be tested to see if their features are being utilized correctly. For this server, extrude, sketches, revolves, and boolean operators will be supported to test the integration

1.4 **Document Organization**

In the next chapter, an overview of the current research done in this area will be presented and how this research adds to the various technologies. Chapter three will present the method and describe the road map for creation of a complete MMORPG architecture. Chapter four will provide an explanation of how the architecture was implemented into the current architecture utilized by NX Connect. Chapter five will present the results of the implementation and describe in detail how this new architecture reduced bandwidth consumption as well as demonstrates the scalability of this architecture. Chapter 6 will present suggestions for future work.

2 BACKGROUND

The research that is presented builds on the work of other collaborative, multi-user projects as reviewed in the following sections. Tools like ROCCAD, CADDAC, and NX Connect have been developed for use in engineering but have some architectural deficiencies in how they are created. A review of the literature will be given to familiarize the reader with the work that has been done with these collaborative tools as well as their architectures. The review begins with an overview of current architectures and how multi-user tools have utilized them. This is followed by an explanation of MMORPG architectures, specifically their creation, development, and usage. These reviews will explain the current advantages and disadvantages of each architecture and demonstrate the need for an implementation of a scalable, efficient architecture like those found in MMORPG client-server architectures. During the review, the relationship of prior research and how it relates to the MMORPGs will be emphasized.

2.1 **Multi User Architectures**

Multiple architectures exist in practice, including client-server, peer-to-peer, and MMORPG client-server architectures, each offering a unique method to pass and store data. Understanding how these architectures differ and function is necessary to see how these can be successfully utilized but also show their deficiencies. It is therefore important that their frameworks be explored for both peer-to-peer (P2P) and client-server (CS) architectures so the value and contribution of the methods presented in this research can be understood.

2.1.1 Peer-to-Peer Architectures

Peer-to-peer (P2P) architectures store information in the clients, and multicast the data to their peers. Multicasting involves packaging packets of information that are required by each tool and sending it to a peer that is connected to a single computer, which eventually gets propagated to every computer in the network. Each computer can act as both client and server. The advantage to this is that it is "convenient since the users can form a design group by connecting directly to each other anytime anywhere without the presence of a central server." (Chen, 2007) This is useful because it removes the overhead cost of hardware needed to maintain a server.

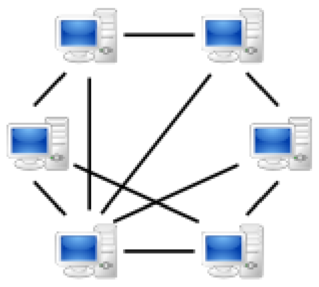


Figure 2-1: Peer to peer architecture

Also, if a single client has a malfunction, that client can leave the project without affecting other peers. These connections that form a network (see Figure 2-1) also give the ability to create a load balancing system for handling the various messages that are passed. This load balancing system has to be created for each P2P framework. Ideally, the load balancing framework allows the processing power of all of the connected clients to be uniform throughout the network. However, the load balancing system is also limited by the amount of processing power that the combined clients individually hold and the network can become overwhelmed because of the slow processors of an individual or multiple clients, generating lag or delayed reception of dataA significant amount of lag can occur if intense computations are needed like those found in most CAx tools. Also, a “lack of an established IP Multicast solution forces such architectures to consume a lot of bandwidth” (Assiotis, Marios, Velin, 2010), making it more consuming to pass messages when there are multiple clients attached to the network. One example of an engineering tool that utilizes this framework is ROCCAD. In ROCCAD (Chen, 2007), messenger systems were set up to send information about CAD models from one peer to another. As each client connected to the framework, it was able to receive messages and also send its own messages. The communication system that is utilized can help pass the messages, but the rendering and the model data are all done on the client.

2.1.2 Client-server Architectures

Client-server architectures are advantageous in that all of the calculations are handled by a central hub. This is beneficial because the server can handle lookup of users, design groups on the web, and provide extra functionality such as maintaining a master copy of the model and back up of models for design changes (Chen, 2007).

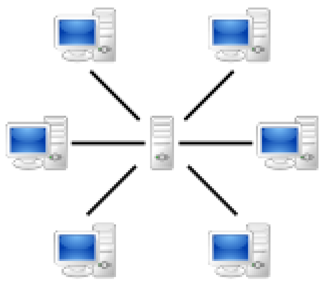


Figure 2-2: Client-server architecture

Having a centralized system (see Figure 2-2) allows users to know where and what they are connecting to. It also stores a master copy of work done in a centralized area, with backup capabilities built in system.

Architectures with a single server do experience synchronization issues. "The major bottleneck for the client-server architecture is its ability to handle communication at the server end as most data and resources are located at the server. Since clients cannot directly access the resources at the other clients, the server has to process several jobs and pass the data from one node to another node, which burdens the system and slows down the network." (Fan et al. 2010). Any server is limited by its own hardware. With a single server at the heart of the architecture, intense computations can also bog down the system with an increase in the number of clients, depending on how much the client handles and what type of data is passed over the network. With CAx applications, especially Finite Element programs, a significant amount of data and calculations are processed that require a significant amount of processing power. These types of programs, when expanded into collaborative, multi user applications, can consume a great deal of processing power and bandwidth for a single server.

Some examples of engineering tools that use this architecture include CADDAC, NX Connect, and CATIA Connect. In CADDAC (Ramani, 2003), the server itself holds a kernel that does all of the geometry computations. All of the geometry, constraints, session management, data management and synchronization of clients are handled on the server (Ramani). This allows clients to keep their side fairly light, only requiring the actual rendering of the data. NX Connect (Ryskamp, 2010) and CATIAConnect both are plugins to current existing commercial single user CAD applications which run on a similar architecture, providing a multi-user experience when run. They send any data from the client to the server where a master model is created. The data are then rendered on the other clients logged into the server using the data stored in the master model.

2.2 **MMORPG's**

Massive Multiplayer Online Role Playing Games (MMORPGs) have been around since 1997 (History of Massively Multiplayer Online Games, 2007) . Historically, Massive multi-player online games (MMO's) and MMORPGs were built using a simple client/server architecture. (Dickey, 2004). Today you can find some games using peer-to-peer environments and clusters (network of servers). Most known MMORPGs that exist today are World of Warcraft, EVE Online, and SecondLife, each of which has a unique MMORPG architecture that utilizes a network of servers (Figure 2-3). These networks of servers are able to load balance up to several thousand users at a time, at the cost of supplying the physical machines on which the servers operate. As games become more complex and integrate physics engines and mimic real life graphics, this architecture is able to spread the computation load across multiple servers so that the end user experiences little to no lag, even if thousands of users are logged into the system. In the gaming environment, when a user moves from one virtual gaming world to another virtual world, the screen will black out for a minute while the virtual data is loading. After loading the new world will be displayed, and then the user will see what is on the other end of the tunnel, cave, or means of passage. During this delay period data is being transferred from one server to another, both running the same instance of the virtual world. With multi-user tools, features are passed to the server and collected into a single model or assembly or mesh, depending on the type of CAx tool. This data can similarly be transferred to other servers for analysis or split into multiple projects.

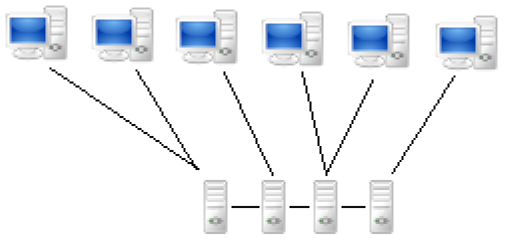


Figure 2-3: Cluster architecture

EveOnline experienced over 65000 users logged in at one time (CCP Games, 2010). Research is continually ongoing to improve the MMORPG architecture, including development of a fully distributed MMORPG architecture (Assiotis, 2005) and architecture for a MMORPG game engine (Caltagirone, 2002). To see the advantages of MMORPG servers over P2P and client-server architectures, some background information is presented on what they are used for, what they can be applied to, and how they work.

2.2.1 Creation

Originally, computer games were single player or multi-player turn based games, which did not require specific network architecture to run. Most games could be run on a single computer and little communication was performed with users on other computers at other locations. “Faster networks, faster processors, and 3D accelerator cards have contributed to the push for a new genre of online games” (Caltagirone, 2002). These games required an increase in processing power, reliability, scalability, and performance. Since games make up a huge portion of the computer industry, the push to advance hardware has helped computers become more advanced so they could support these types of games. As these games began to catch a lot of attention, more protocols were introduced to keep track of users logged in, to keep the servers secure, to keep data backed up, and to keep the virtual gaming world persistent on the server.

2.2.2 Development

As MMORPG’s came into being, extra services became necessary to maintain the game integrity and to keep unregistered users out of the system or from cheating. Caltagirone outlined six goals to creating a secure, stable, fully distributed architecture. The requirements of such a system are to “minimize network traffic, provide …load balancing, secure gaming environment, a high level of scalability and maintainability, and …client side performance for real-time graphics.” (Caltagirone, 2002) Each game development company has a set of similar goals to create a stable system with similar services. Nevrax, a software game application company that was bought out by Winchgate, developed a MMORPG called Ryzom which is now open source. This game utilizes NEL, a development toolkit for the development of client and server side technologies. Ryzom has a list of services that it utilizes to spread the computation load of the game across multiple physical machines (Lang, 2011). These services can be utilized to spread the computational load across multiple servers, handle backup management, manage AI for characters in the game, and send messages back and forth to hundreds and thousands of clients logged into the architecture. IBM also utilizes several of the same ideas and is developing a high level MMO architecture that has similar services, albeit implemented differently (Chu, 2008).

2.2.3 Usage

Cluster architectures are currently used only for online role playing games that require lots of graphical rendering and where the game needs to track information of the user. They have been used for studies to simulate real world scenarios to predict certain behaviors of various diseases and population growth (PC Zone, 2005), but as of yet this architecture has not been utilized for engineering software tools.

2.3 **Foundational Tools**

Some of the tools that were used to create this prototype have proven necessary to communicate with the current tools that are used. A brief overview of these tools is presented in the following sections.

2.3.1 Foundational Tools: NEL

A messaging system that could carry messages from the client to the database was needed. This is turn also needed to push messages out to clients that were connected to the system. NEL was chosen as this messaging service, as it is currently being implemented in MMORPGs. It is a development kit for client and server technologies for MMORPGs and provides basic an application programming interface (API) that is capable of utilizing push technology and also spreading the network services across multiple physical machines. It is written in C++ and is currently used for the MMORPG Ryzom (Ryzom, 2006).

2.3.2 Foundational Tools: XML

Developing this architecture requires a format to pass messages in. Since the database is structured to be feature based, each feature needed a unique message to be passed. XML is a simple format that can be used online and is widely used in arbitrary data structures like web services. It is compatible with SOAP, as well as hundreds of other languages. Its format is simple text, making it easier to serialize and de-serialize.

2.3.3 Foundational Tools: SOAP

Simple Object Access Protocol, or SOAP, is a protocol for exchanging information in computer networks. It works very well with XML and can be used to serialize and de-serialize XML text. It generally used HTTP for its application layer. It also easily can communicate through firewalls and proxies. NX Connect is written in C# and will be used to test this architecture. SOAP can convert the needed messages from NEL (written in C++) to NX Connect using the appropriate format calls.

3 METHOD

This chapter describes the method for the creation and integration of MMORPG architecture. The integration will merge the server created with any existing engineering tool. To assist with clarification and provide context, some examples have been included with illustrations. For specific implementation of the architecture created, refer to Chapter 4.

3.1 Overall Method

The method for creating and integrating a MMORPG client-server architecture consists of several parts (Figure 3-1). The most general steps include:

1. Creating services for client to server, server to client, and server to server communication.
2. Create a client that can communicate with the services.
3. Create a service to database communication system.
4. Integrate the new server built on services with a multi-user engineering tool.

While these steps seem simple and might lead the reader to wonder whether or not the architecture described can actually reach the proposed research objectives, the reader should note that this server architecture has already been implemented successfully for several MMORPGs. The computer architecture easily stores the necessary data required to create the necessary communication pathways with limited bandwidth consumption and for this reason it was chosen to be implemented with collaborative multi-user engineering tools.

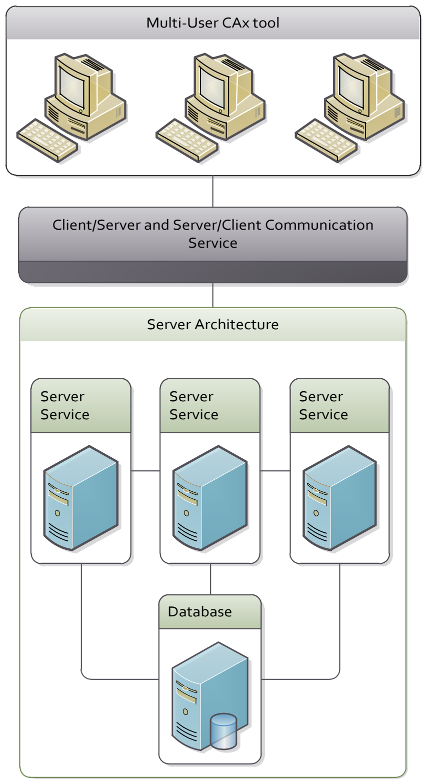


Figure 3-1: Layout of necessary communication connections

3.2 **Creating Services**

As mentioned earlier, for any MMORPG architecture a set of services are required to handle the communication between the client and the server. These services in a game can include management of non-human characters, backup management, cluster administration, position of characters, login information, synchronization services, etc. All of these services are intended to provide the six goals outlined by Caltagirone (Caltagirone, 2002):

1. Minimize network traffic.
2. Provide load balancing.

3. Create secure gaming environment.
4. Give a high level of scalability.
5. Maintain architecture easily.
6. Keep client light to increase graphics performance.

For the purpose of this research, the goals have changed slightly. Since the architecture is to be built for the purpose of utilizing engineering tools, there is no gaming environment. Because of this, the only data that needs to be stored are the features created inside the various CAx tools. The goals then become:

1. Minimize network traffic
2. Provide load balancing.
3. Give a high level of scalability for small to large size CAx tools.
4. Maintain architecture easily.
5. Keep client light to increase overall performance.
6. Provide extensibility.

It should be noted that to provide load balancing for the server, backend management services would be required which would provide services such as load data management, back up services, and load balancing. Nel provides API functionality to create such services, but these services are beyond the scope of the project and will not be tested now but details are included in Chapter 6.

By creating services that provide these basic abilities, the cluster server can be created to distribute computation loads and allow all messages to be propagated to any connected clients. The client listeners can then be turned on, thereby decreasing communication bandwidth by creating a server-push client model. The method does not prescribe what format or language should be selected to create these services. As described in Chapter 4, utilizing an existing tool like NEL can be an excellent option for creating the desired services. However, it is not necessarily required to use NEL. A variety of open source game servers can be found on the internet or a commercial package can be purchased. A custom game server can be created by creating a variety of services with communication pathways between services, but this can be time consuming and expensive. As long as the listed requirements are met, any option to create the services would be acceptable and the architecture can be used, meeting the objectives of this research.

3.3 **Client Communication from Tool to Server**

To effectively send data from the multi-user engineering tool to the server, a communication protocol is required to communicate between them. If the multi-user engineering tool is written in a language that is compatible with the server, communication can occur directly via TCP/IP, HTTP, or UDP. However, if the user does not have direct control over what language the game server or engineering tool are written in, an intermediary plug-in can be created that communicates between the server services and the engineering client. Since this is more often the case and more difficult to implement, an example of this was used in the implementation found in Chapter 4. There also is needed a client for the multi-user engineering tool to communicate with in order to send information from the multi-user tool to the server. This was accomplished in Chapter 4 by creating a client .DLL file that takes messages and passes them to and from the server. SSL or some other security protocol should be utilized to encrypt the data being transmitted across communication pathways from the tool to the server, which will be discussed in Chapter 6 for future work.

For this method, the features are passed back to the connected clients via the Game Client, but the attached tool can be anything. Similar to the method of neutral command calls (Min, 2007), the objects that are passed from the MMORPG server to the client can be made platform independent. What is utilized in one CAx tool can be sent to another via the MMORPG server. Like the command neutral CAD tool, however, the code used to pass information must also be updated with changes to any CAx API.

3.4 **Server to Database Communication**

The features created in a CAx engineering tool are sent to all the clients connected to the server, and the server needs to propagate all changes of the model to the database and to the clients. The services created inside of the server using a game server development kit can be used to propagate the necessary information to both CAx collaborative tools and database through the server, maintaining a single master copy of the needed data. This service needs to also propagate information back to the clients if any exceptions or database insertion failures occur to notify all users that the feature passed failed to get inserted into the master copy.

4 IMPLEMENTATION

This chapter presents the overall methodology described in the previous chapter on how to create services for the server portion of the architecture, how to create and connect a client, and how to integrate the architecture with an existing multi-user tool. The end result of this research is CAx Connect, a MMORPG type client server architecture (Figure 4-1). This architecture is integrated with NX Connect, a plug-in for the Siemens UG NX 6.0 CAD package. This architecture will enable the engineering tool to overcome some of the obstacles it inherently created when it was developed. The main obstacle concerned with this research is reducing bandwidth consumption.

While this architecture does not completely handle many of the aspects of a fully developed MMORPG architecture, it does demonstrate the usefulness of creating and integrating this architecture. Data needs to be stored within the architecture using a database. The backend for this project is a Microsoft SQL Server 2008 database, which interfaces nicely with C# using Linq. For information about what Linq is and what it does, go to http://msdn.microsoft.com and search for Linq. To show how to get started with the implementation, an overview of the setup of NEL is first presented (**Error! Reference source not found.**).

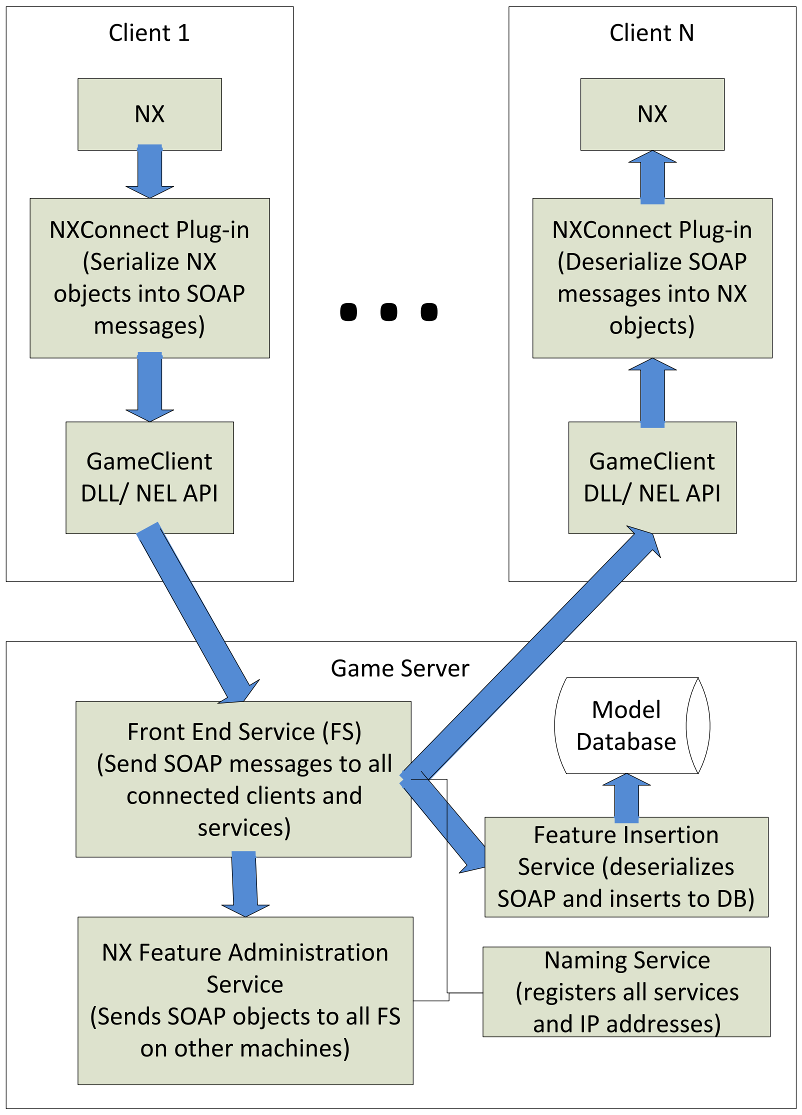
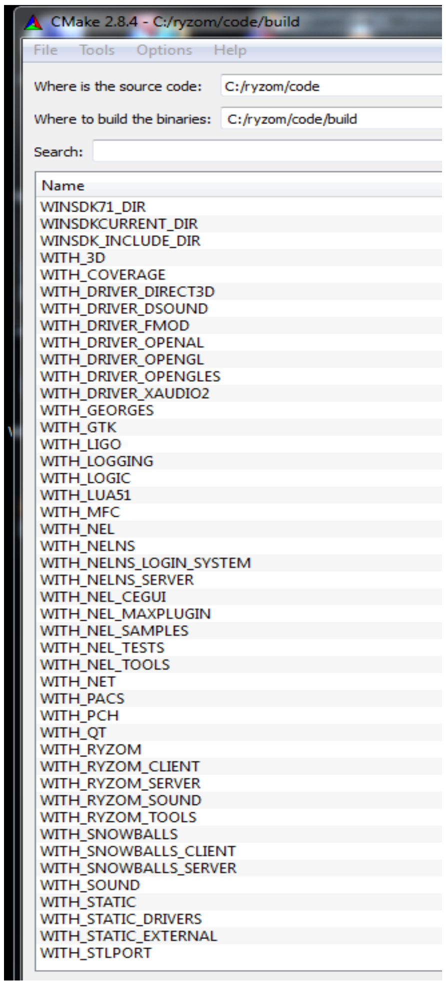

Client 1
NX
NXConnect Plug-in (Serialize NX objects into SOAP messages)
GameClient DLL/ NEL API
Client N
NX
NXConnect Plug-in (Deserialize SOAP messages into NX objects)
GameClient DLL/ NEL API
Game Server
Front End Service (FS) (Send SOAP messages to all connected clients and services)
Model Database
Feature Insertion Service (deserializes SOAP and inserts to DB)
NX Feature Administration Service (Sends SOAP objects to all FS on other machines)
Naming Service (registers all services and IP addresses)

Figure 4-1: CAx Connect architecture

4.1 Overview of NEL and Setup

NEL is an open source tool kit used for the development of MMORPG universes. It contains the source code for writing services for the server and code for the client where a game can be built. The source code for NEL is written in C++ and can be obtained following the instructions found at dev.ryzom.com (see Appendix A). CMake, an open source compiler, can be used to compile the code into a format that any programming interface can use. Visual Studio 2008 was chosen for this research because NX Connect is written in C#, which makes integration with this tool somewhat convenient. There is a list of other prerequisites and instructions found by clicking on the "Build Source CMake (Windows) link. When compiling with CMake, there are a set of options that need to be selected to include the appropriate source code. These options, if all selected, will build the MMORPG Ryzom game that NEL was intended for. To get the source code for just the NEL toolkit, the following options in **Error! Reference source not found.** should be selected which include:



Where is the source code: C:/ryzom/code

Where to build the binaries: C:/ryzom/code/build

Search:

Name
WINSDK71_DIR
WINSDKCURRENT_DIR
WINSDK_INCLUDE_DIR
WITH_3D
WITH_COVERAGE
WITH_DRIVER_DIRECT3D
WITH_DRIVER_DSOUND
WITH_DRIVER_FMOD
WITH_DRIVER_OPENAL
WITH_DRIVER_OPENGL
WITH_DRIVER_OPENGLES
WITH_DRIVER_XAUDIO2
WITH_GEORGES
WITH_GTK
WITH_LIGO
WITH_LOGGING
WITH_LOGIC
WITH_LUA51
WITH_MFC
WITH_NEL
WITH_NELNS
WITH_NELNS_LOGIN_SYSTEM
WITH_NELNS_SERVER
WITH_NEL_CEGUI
WITH_NEL_MAXPLUGIN
WITH_NEL_SAMPLES
WITH_NEL_TESTS
WITH_NEL_TOOLS
WITH_NET
WITH_PACS
WITH_PCH
WITH_QT
WITH_RYZOM
WITH_RYZOM_CLIENT
WITH_RYZOM_SERVER
WITH_RYZOM_SOUND
WITH_RYZOM_TOOLS
WITH_SNOWBALLS
WITH_SNOWBALLS_CLIENT
WITH_SNOWBALLS_SERVER
WITH_SOUND
WITH_STATIC
WITH_STATIC_DRIVERS
WITH_STATIC_EXTERNAL
WITH_STLPORT

Figure 4-2: NEL CMake options

- WITH_EXTERNAL
- WITH_NELNS
- WITH_NEL
- WITH_NEL_SERVER
- WITH_NET
- WITH_STATIC
- WITH_STLPORT

When the solution is generated for CMake, all of the code will be available for use and services can start to be developed.

4.1.1 Creating NEL Services

Services in NEL are atomic. Atomic, in this sense of the word, relates to atomicity, which applies more to the multi-threading processes that it does here. However, it will be used in this context as a way to describe a process that is indivisible. The services that are created are all executable files and pass messages back and forth between services. They need to all be running, or nothing occurs. This is important because the server, consisting of a set of services, will only work if all services are running. Development of new services also requires that each

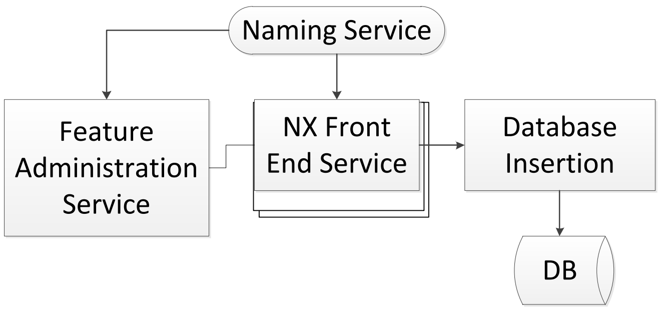


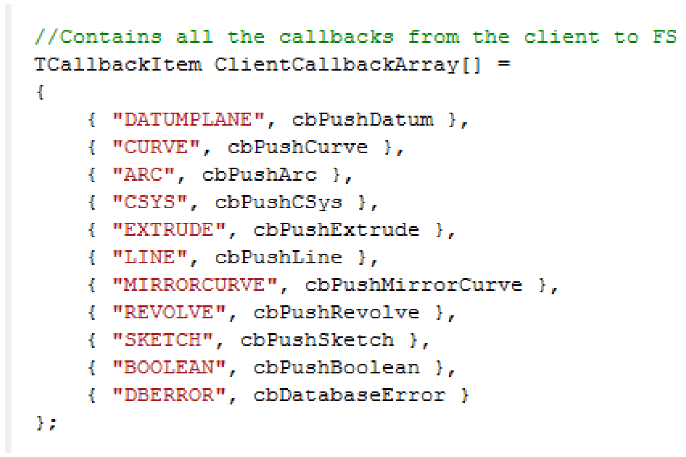
Figure 4-3: Flowchart of NEL services

new service be atomic. There are some templates that are provided to develop new services that are atomic. The flowchart in Figure 4-3 illustrates how the flow of the services works. The first service to come online is the NEL naming service (NS) which is part of the NEL source code and does not require any modifications. The purpose of the NS is to dynamically assign internet protocol (IP) address to the services when they come online. The front end service (FS) contains NX specific code, which can be modified to include code for any features used by other multi-user CAx applications. The feature administration service (FAS) passes messages from the front end service to all other front end services within the network. When turned on, both FS and FAS services register with the NS so that the NS can give each other service the needed IP address to send messages back and forth, acting much like a phone directory. Multiple FS can be run on multiple physical machines, which all can receive features from a client by sending a feature to the FAS which propagates messages to all FS.

The code for this research is specific to NX Connect, as each feature that is created must have an associated tag to go into the CAx Connect call back array. However, this code can be added to or modified to support any multi-user CAx tool, including CATIA Connect or Cubit Connect, for example, which are both multi-user CAx tools that provide asynchronous and synchronous capabilities. Similarly, the information can be passed from a multi-user CAx tool like NX Connect to the server, and then can be passed back to a client that might be running a different multi-user CAx tool like CATIA Connect. The MMORPG architecture provides language agnostic capabilities but has the same API limitations found in research done on neutral command multi-user CAD (Min, 2007).

4.1.2 Getting the Templates

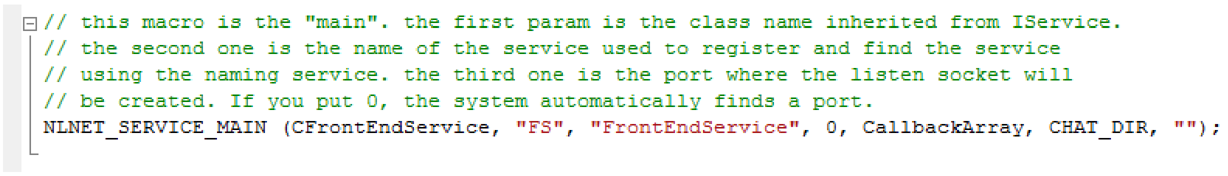
Inside the source code are a set of templates that can be used to create a client and server. They can be found inside the main\code\nel\samples\net\chat folder (i.e. if the main folder is C:\Ryzom, then the templates are located in: C:\Ryzom\code\nel\samples\net\chat). The services and client code are available to create a ground work. The client only sends messages to a front end service (FS) which then relays messages to the appropriate services. The FS utilizes a thread call back (Figure 4-4) to call the corresponding functions to send the message to the correct destination. For every feature to be passed through NEL services, there needs to be a corresponding function.



```
//Contains all the callbacks from the client to FS
TCallbackItem ClientCallbackArray[] =
{
    { "DATUMPLANE", cbPushDatum },
    { "CURVE", cbPushCurve },
    { "ARC", cbPushArc },
    { "CSYS", cbPushCSys },
    { "EXTRUDE", cbPushExtrude },
    { "LINE", cbPushLine },
    { "MIRRORCURVE", cbPushMirrorCurve },
    { "REVOLVE", cbPushRevolve },
    { "SKETCH", cbPushSketch },
    { "BOOLEAN", cbPushBoolean },
    { "DBERROR", cbDatabaseError }
};
```

Figure 4-4: Thread callback array

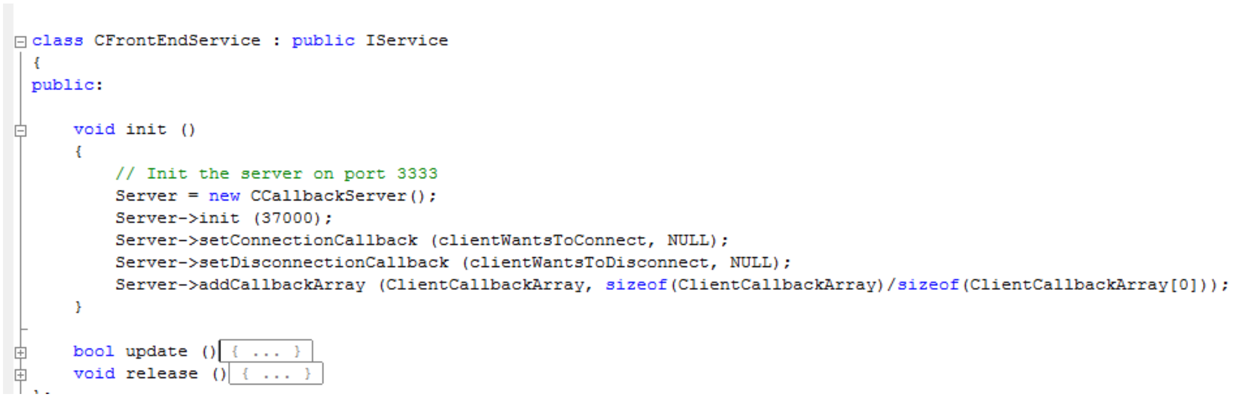
For this research, the FS and a feature administration service (FAS) were created using this template to take feature data from NX Connect and place it into a database. To utilize the callback array, the IService class from which all services derive uses a template for the main (Figure 4-5) function in each service so that the user can specify the array that is to be used. The template requires the class name of the service to be registered, the short name, the long name, the port number, the thread callback array, and then where the directory will be for the last variable, the log file.



```
// this macro is the "main". the first param is the class name inherited from IService.
// the second one is the name of the service used to register and find the service
// using the naming service. the third one is the port where the listen socket will
// be created. If you put 0, the system automatically finds a port.
NLNET_SERVICE_MAIN (CFrontEndService, "FS", "FrontEndService", 0, CallbackArray, CHAT_DIR, "");
```

Figure 4-5: NEL API main template

In order to derive service from the IService class, 3 functions need to be implemented: init, update, and release, which are called sequentially when the service is initialized (Figure 4-6).



```
class CFrontEndService : public IService
{
public:

    void init ()
    {
        // Init the server on port 3333
        Server = new CCallbackServer();
        Server->init (37000);
        Server->setConnectionCallback (clientWantsToConnect, NULL);
        Server->setDisconnectionCallback (clientWantsToDisconnect, NULL);
        Server->addCallbackArray (ClientCallbackArray, sizeof(ClientCallbackArray)/sizeof(ClientCallbackArray[0]));
    }

    bool update (){ ... }
    void release (){ ... }
```

Figure 4-6: Sample code for service derived from IService class

The update function runs on a timer that is created by the CUnifiedNetwork class, which is a singleton, allowing only a single instance during the duration of the program but can be accessed by getting the instance. It is through this singleton that all services can pass messages.

4.1.3 Sending Messages

In order to send messages to each service, the services must each register with a NS which NEL does automatically. The process for sending messages from the multi-user tool to

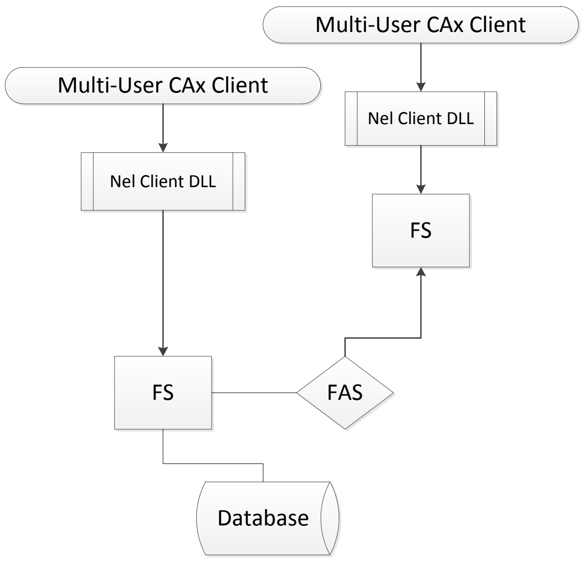
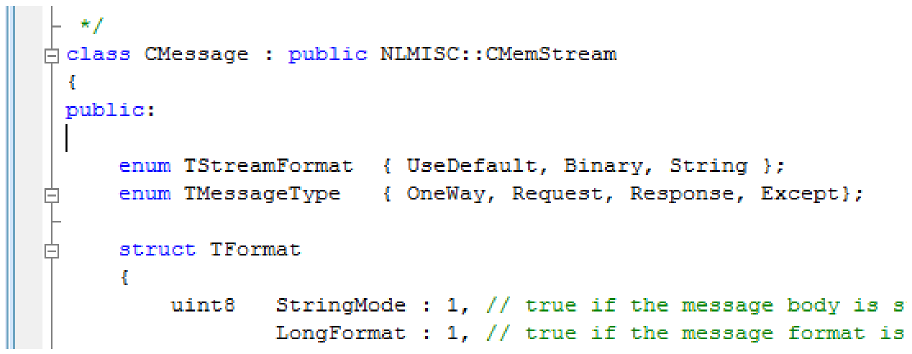


Figure 4-7: Flowchart of message passing

the server and back can be seen in the flowchart in Figure 4-3.

The multi-user tool sends some feature or object to the NEL Client DLL. This DLL then is connected to the FS and passes that feature in a string to the FS. Inside each service, messages are packaged up using a serialize function built in NEL and put into CMessage objects (Figure 4-8), which the services are able to send back and forth.

These objects know which destination to go to because a set of call back arrays are created that include tags and callback functions, as well as an IP address provided by the NS.



```
 */
class CMessage : public NLMISC::CMemStream
{
public:

    enum TStreamFormat  { UseDefault, Binary, String };
    enum TMessageType   { OneWay, Request, Response, Except};

    struct TFormat
    {
        uint8   StringMode : 1, // true if the message body is s
                LongFormat : 1, // true if the message format is
```

Figure 4-8: CMessage declaration

Clients send their messages first to the FS, which then can pass the message along to other services created. The FS will send the CMessage containing the feature to the FAS which will take the message and send it out to all FS that are registered with the naming service, allowing multiple front end services to be run on multiple machines. This message is then passed to the database, where it is inserted and retained as a master copy. The feature is then pushed out to any connected clients.

4.1.4 Callbacks

Each service has its own set of callback functions. NEL utilizes these callback functions to pass CMessage objects from service to service, client to server, or server to client. A callback array is created for the network using TUnifiedCallBackItem while callbacks for the client to server use a TCallbackItem array. These arrays are able can take in two values, a tag for the service to have a message sent to, and a callback function to call when a service receives a message with a specific tag (Figure 4-4). This is what makes the services atomic. These arrays can call functions for all many services, allowing each service to run independent of other services, but also will make nothing occur if the service isn't running.

4.2 Implementation into NX Connect

Once the services are created and a client can connect to the services, the architecture can be implemented into a collaborative multi-user engineering tool. NX Connect was chosen because it is a plug-in that has been proven to be an efficient multi-user CAD system and can be easily modified to connect to the architecture. NX Connect is feature driven, that is, rather than storing part files inside a database it stores the various features that were created inside a specific part. The changes made to NX Connect to integrate with CAx Connect can be shown in Figure 4-9. Any features sent from NX to NX Connect are blocked if the option to use the CAx Connect MMORPG architecture is selected. This diverts normal operation and sends any features to the DLL. This will be discussed more in depth later in 4.2.1 and 4.2.2.

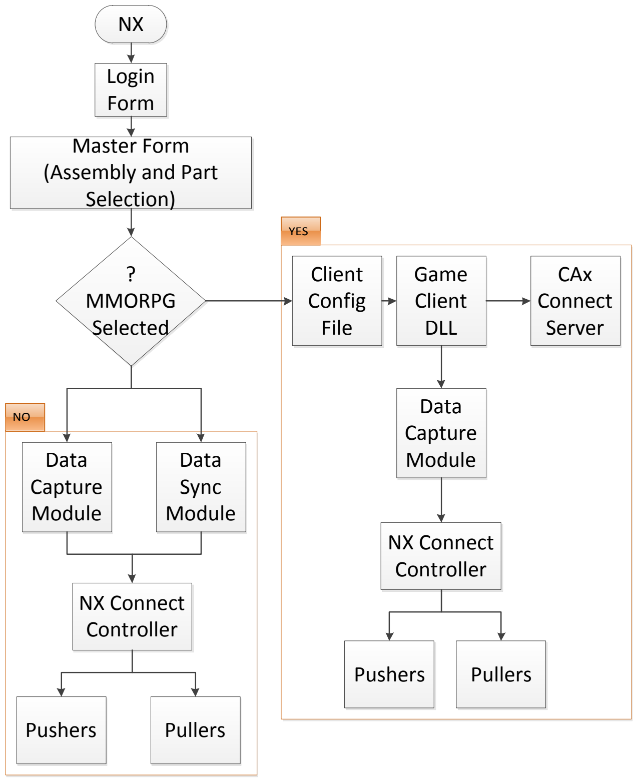
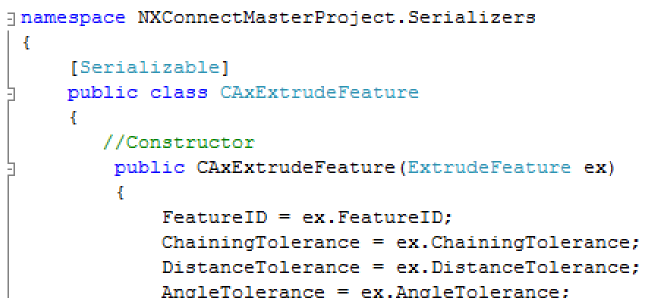


Figure 4-9: Flowchart of modifications to NX Connect

4.2.1 NX Connect SOAP Serialization

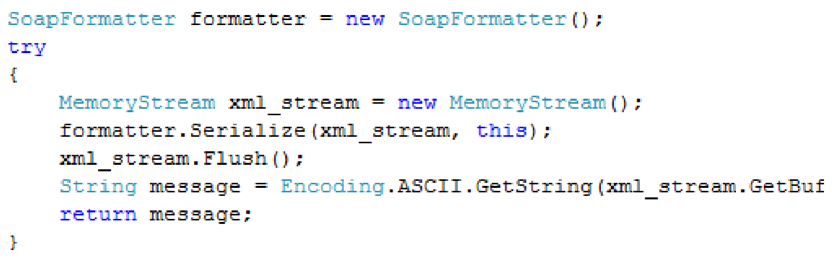
Each feature generated inside of NX and passed to NX Connect needs to be put into a serializable object to pass the feature to CAxConnect (Figure 4-10) which mirrors the database structure object generated by Linq. Linq automatically generates objects that mirror the database architecture designed in Microsoft Windows SQL Server.



```
namespace NXConnectMasterProject.Serializers
{
    [Serializable]
    public class CAxExtrudeFeature
    {
        //Constructor
        public CAxExtrudeFeature(ExtrudeFeature ex)
        {
            FeatureID = ex.FeatureID;
            ChainingTolerance = ex.ChainingTolerance;
            DistanceTolerance = ex.DistanceTolerance;
            AngleTolerance = ex.AngleTolerance;
```

Figure 4-10: Serializable object declaration

Each feature passed has a serializable object that mirrored the Linq objects utilized in NX Connect because Linq objects cannot be serialized into XML format. For every feature to be passed through to CAx Connect, there must be a serializable object. These objects receive the information from NX Open, the API created by Siemens to make custom NX applications. These objects then are put into an xml format and cast to a string (Figure 4-11) using a soap formatter in C#.

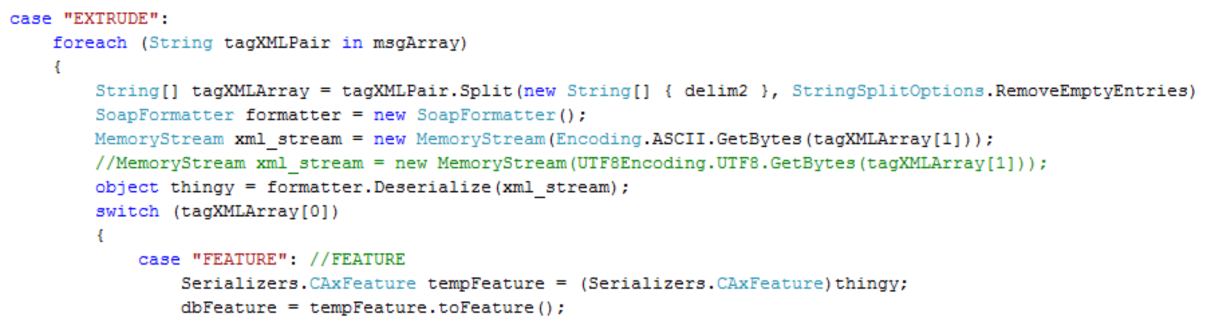


```
SoapFormatter formatter = new SoapFormatter();
try
{
    MemoryStream xml_stream = new MemoryStream();
    formatter.Serialize(xml_stream, this);
    xml_stream.Flush();
    String message = Encoding.ASCII.GetString(xml_stream.GetBuf
    return message;
}
```

Figure 4-11: SOAP formatter code

This string is then sent to the FS via the Game Client DLL with an associated tag that determines what type of feature is being passed.

Likewise, any features created by other clients are propagated back to all users using the Game Client DLL. When a user receives this feature on his or her client, the message must be deserialized and then inserted into an NX Open object (Figure 4-12).



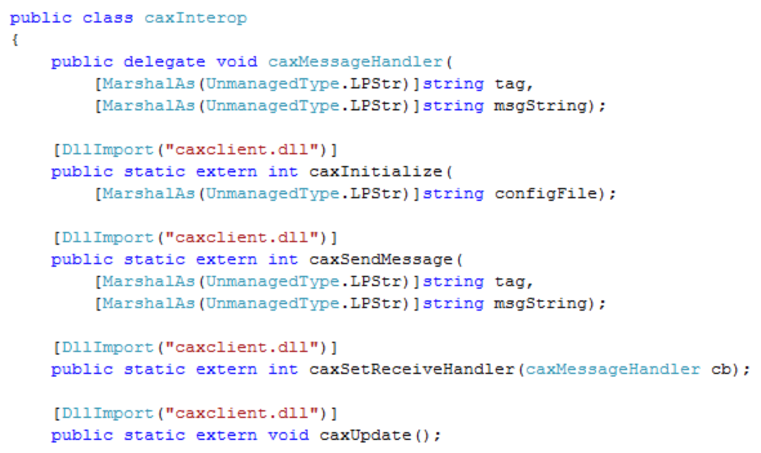
```
case "EXTRUDE":
    foreach (String tagXMLPair in msgArray)
    {
        String[] tagXMLArray = tagXMLPair.Split(new String[] { delim2 }, StringSplitOptions.RemoveEmptyEntries)
        SoapFormatter formatter = new SoapFormatter();
        MemoryStream xml_stream = new MemoryStream(Encoding.ASCII.GetBytes(tagXMLArray[1]));
        //MemoryStream xml_stream = new MemoryStream(UTF8Encoding.UTF8.GetBytes(tagXMLArray[1]));
        object thingy = formatter.Deserialize(xml_stream);
        switch (tagXMLArray[0])
        {
            case "FEATURE": //FEATURE
                Serializers.CAxFeature tempFeature = (Serializers.CAxFeature)thingy;
                dbFeature = tempFeature.toFeature();
```

Figure 4-12 SOAP deserialization code

When the NX Open object is populated by the information contained in the xml string, it then calls a pusher from NX Connect and reconstructs the object on the client side of every user who receives the message from the server.

4.2.2 The Game Client DLL

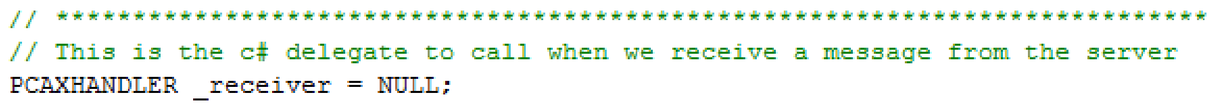
The Game Client DLL is a C++ .DLL file that is marshaled inside of NX Connect and the Feature Insertion Service (FIS) (Figure 4-13).



```
public class caxInterop
{
    public delegate void caxMessageHandler(
        [MarshalAs(UnmanagedType.LPStr)]string tag,
        [MarshalAs(UnmanagedType.LPStr)]string msgString);

    [DllImport("caxclient.dll")]
    public static extern int caxInitialize(
        [MarshalAs(UnmanagedType.LPStr)]string configFile);

    [DllImport("caxclient.dll")]
    public static extern int caxSendMessage(
        [MarshalAs(UnmanagedType.LPStr)]string tag,
        [MarshalAs(UnmanagedType.LPStr)]string msgString);

    [DllImport("caxclient.dll")]
    public static extern int caxSetReceiveHandler(caxMessageHandler cb);

    [DllImport("caxclient.dll")]
    public static extern void caxUpdate();
```

Figure 4-13: Interop code marshaling .DLL

It allows the passage of a string constructed in C# to pass through C++ code. When an instance of the client is created using an interop inside of NX Connect, the instance searches for a configuration file to notify the client where the naming service is. When it goes to the specified IP address, it will be notified of all the IP addresses of all front end services connected to the naming service. It will then be assigned to a front end service, dynamically connected by the naming service so that one front end service is not burdened unnecessarily. The .DLL contains code to allow messages to be sent to CAxConnect from C# code by creating a function



```
// *************************************************************************
// This is the c# delegate to call when we receive a message from the server
PCAXHANDLER _receiver = NULL;
```

Figure 4-14:Delegate instantiation used as a function pointer

pointer to a handler that is contained inside of NX Connect and the database insertion service. When a message is propagated through NX Connect to the DLL, the send message call is made and the server receives the message. When the server needs to send messages back to the clients, it utilizes a similar call back array as seen in Figure 4-4 and then calls the function with the pointer to the receiving end. The receiving end, contained in NX Connect and the Feature Insertion Service, can then do whatever is specified in the function. For NX Connect, the object is deserialized, and then reconstructed into an NX Open object and rebuilt on the client.

4.2.3 Insertion into the Database

When implementing this into any multi-user tool, the tool will eventually need to pass the information needed to some data structure in order for other users to access that information. When that information is passed, rather than directly connecting to the data structure, it can be passed to the FS, which will relay the message to the appropriate service. The (FIS) was created using C# (Figure 4-1) which contains a receiving method that is marshaled inside of the Game Client DLL. This function is called when it receives a message from the FS and it then takes the message, deserializes the message from a string using the SOAP deserialization method, and then inserts the message into a serializable object that is defined in the NX Connect code. This service is built on the .NET framework, so it utilizes the same .DLL that NX Connect is built on, giving access to the serializable objects. When the message is put into a serializable object, that object is cast to a Linq object and then the feature is inserted into the database. If there is a problem with insertion, the insertion is called inside of a try/catch statement, which will send the error (utilizing the Game Client DLL function pointer) back to all connected clients to the FS.

5 RESULTS AND FUTURE WORK

To verify that the proposed research accomplishes the two objectives, some analysis was performed on the implementation described in Chapter 4. First, an examination of the bandwidth consumption of CAx Connect compared to NX Connect was performed. Second, a review of the architecture and its capabilities is presented and a roadmap was created to outline the future work of this architecture. Based upon this information, the following sections in this chapter present the results of the implementation of the method discussed in Chapter 3.

5.1 Bandwidth Results

To the knowledge of the author, application of this type of architecture for use in multi-user engineering tools has never been done prior to the implementation of CAx Connect. However, in the NX Connect multi-user tool, clients consistently asked for changes to the database by calling a pull function which was attached to a timer. As n users connected to the client server architecture of NX Connect, n amount of bandwidth was consumed every interval on the timer. To create an equation to determine conservative estimates for how much bandwidth is consumed based on how long it takes for a client to send a feature to the database and propagate it to the other users connected:

- The number of users connected to a server = n
- The amount of bandwidth consumed each interval = b (Mbps)

- The number of intervals before a feature is sent from the client to the server and back to the other clients = h
- The amount of bandwidth to send a feature to the database = d
- The total amount of bandwidth consumed = T

Using the variables above, the following equation is created:

$$T=n(bh+d) + d \tag{5.1}$$

This is the case because for every feature sent to the database, the server must propagate those changes to each user connected.

While this proved to be an effective solution for handling multiple users, the amount of users the server could hold was limited by the capabilities of a single server. If hundreds and perhaps thousands of users would like to log into the architecture to utilize the tool, the amount of traffic would reduce the amount of available bandwidth by n users. Similarly, to provide an estimate of the amount of bandwidth consumed, some variables need to be defined:

- The number of users connected to a server = n
- The number of front end services running = f
- The amount of bandwidth consumed per feature sent to the database = d
- The total amount of bandwidth consumed = T

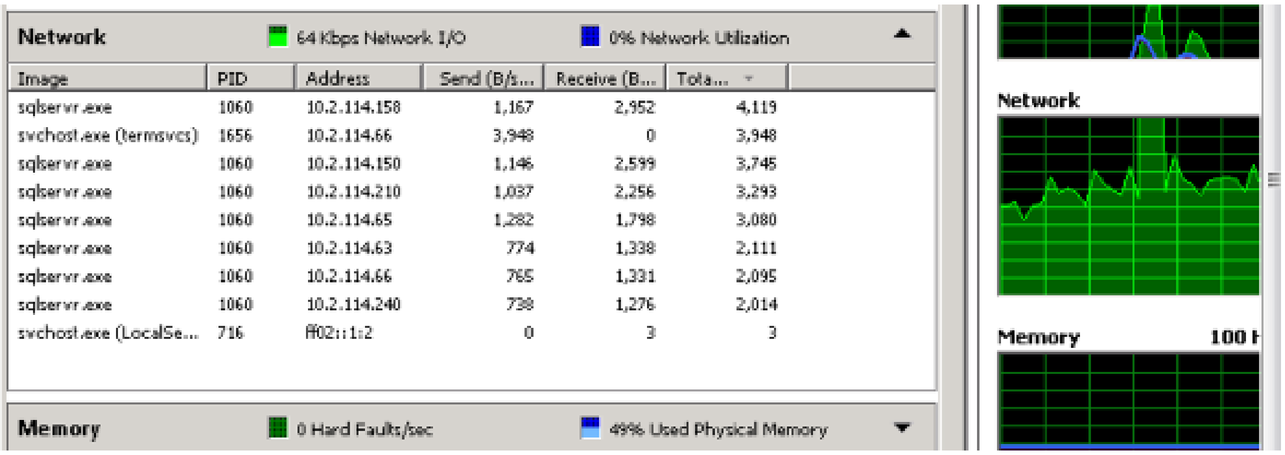
Using the variables above the following equation is constructed:

$$T=(n*d)/(f) \tag{5.2}$$

All clients connected also passed feature data directly to the database, and so some users needed to wait for an available thread. As a result, NX Connect experienced greater lag between users as more clients connected.

CAx Connect allows multiple threads to be sent to the database via the FS. NEL can run multiple instances of a FS and automatically provides load balancing of the number of clients to the amount of available FS's running. As a result, bandwidth is only consumed when a feature is created, not when a timer reaches an interval. The FS can be located on multiple machines as well, increasing the amount of available bandwidth by n number of FS's. This limits the bandwidth of users much less than NX Connect does, and the limit is only based on the number of servers connected to the cluster.

Using the networking tool in the task manager on the server to monitor the bytes received from all connected clients, the bandwidth consumption was measured between the server and the current NX Connect implementation, as well as the server and the CAx Connect implementation. A measurement of the number of bytes sent and received was performed using the task manager



Network 🟩 64 Kbps Network I/O 🟦 0% Network Utilization

Image	PID	Address	Send (B/s...	Receive (B...	Tota...
sqlservr.exe	1060	10.2.114.158	1,167	2,952	4,119
svchost.exe (termsvcs)	1656	10.2.114.66	3,948	0	3,948
sqlservr.exe	1060	10.2.114.150	1,146	2,599	3,745
sqlservr.exe	1060	10.2.114.210	1,037	2,256	3,293
sqlservr.exe	1060	10.2.114.65	1,282	1,798	3,080
sqlservr.exe	1060	10.2.114.63	774	1,338	2,111
sqlservr.exe	1060	10.2.114.66	765	1,331	2,095
sqlservr.exe	1060	10.2.114.240	738	1,276	2,014
svchost.exe (LocalSe...	716	ff02::1:2	0	3	3

Memory 🟩 0 Hard Faults/sec 🟦 49% Used Physical Memory

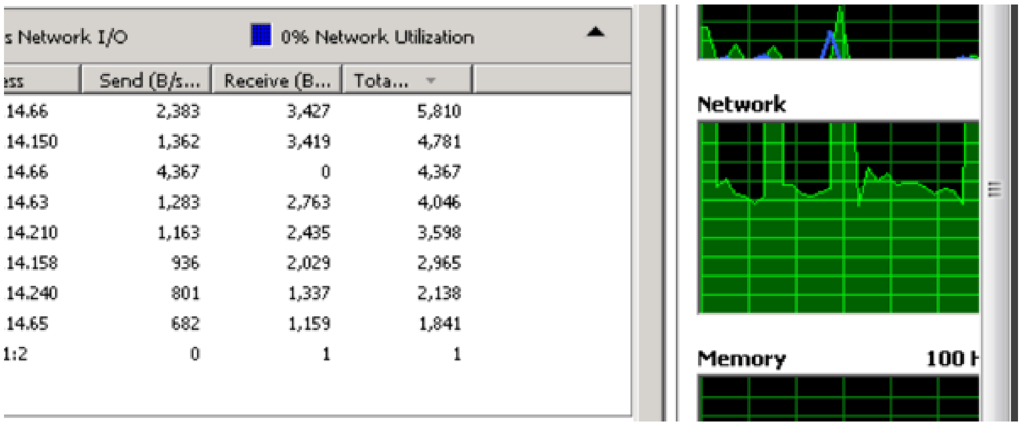
Figure 5-1: Network tool tracking bytes sent/received using current NX Connect

and the data are recorded for each case. 10 users were put on 10 individual machines and each user logged into NX Connect and CAxConnect and performed the same operations as specified by a story board created for this test. The results are shown in Table 5-1.

Table 5-1: Network tracking results

Network Tracking Results	
Average bytes received by server using NX Connect implementation	1935 bytes/sec
Average bytes received by server when feature sent using NX Connect implementation	2367 bytes/sec
Average bytes received by server using CAxConnect implementation	156 bytes/sec
Average bytes received by server when feature sent using CAxConnect implementation	201 bytes/sec

These results were obtained from evaluating data recorded during the test (Figures 19, 20 21 and 22). As can be seen from Figure 5-1, NX Connect with its current implementation utilizes resources periodically, whether features are being passed or not, because NX Connect must continually poll the database for changes inserted by other users.

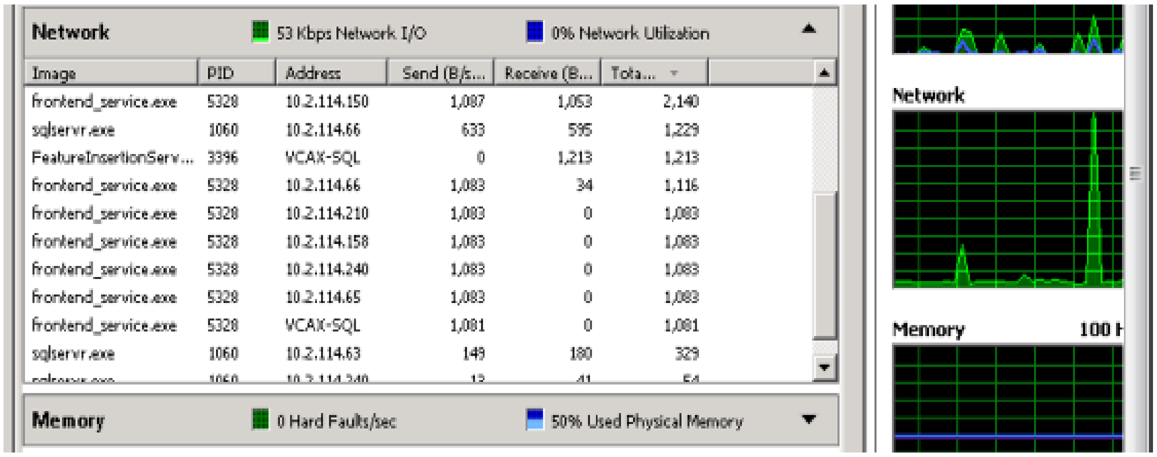


s Network I/O ■ 0% Network Utilization

ss	Send (B/s...	Receive (B...	Tota...
14.66	2,383	3,427	5,810
14.150	1,362	3,419	4,781
14.66	4,367	0	4,367
14.63	1,283	2,763	4,046
14.210	1,163	2,435	3,598
14.158	936	2,029	2,965
14.240	801	1,337	2,138
14.65	682	1,159	1,841
1:2	0	1	1

Figure 5-2: Network tool monitoring feature sent using NX Connect

Whenever a user passed a feature to the server, the network monitoring tool picked up a spike in the number of bytes received. In the case of the NX Connect implementation, this resulted in an increase of nearly 400 bytes per interval.



Network 🟩 53 Kbps Network I/O 🟦 0% Network Utilization

Image	PID	Address	Send (B/s...	Receive (B...	Tota...
frontend_service.exe	5328	10.2.114.150	1,087	1,053	2,140
sqlservr.exe	1060	10.2.114.66	633	595	1,229
FeatureInsertionServ...	3396	VCAX-SQL	0	1,213	1,213
frontend_service.exe	5328	10.2.114.66	1,083	34	1,116
frontend_service.exe	5328	10.2.114.210	1,083	0	1,083
frontend_service.exe	5328	10.2.114.158	1,083	0	1,083
frontend_service.exe	5328	10.2.114.240	1,083	0	1,083
frontend_service.exe	5328	10.2.114.65	1,083	0	1,083
frontend_service.exe	5328	VCAX-SQL	1,081	0	1,081
sqlservr.exe	1060	10.2.114.63	149	180	329

Memory 🟩 0 Hard Faults/sec 🟦 50% Used Physical Memory

Figure 5-3: Network tool tracking bytes sent/received using CAx Connect

CAx Connect similarly was tested by 10 users, performing the same set of operations using the same storyboard. As can be seen in Figure 5-3, these users (frontend_service.exe with associated IP address) utilized 0 bytes per interval as expected. The results monitored indicate a slight deviation for 10.2.114.66, as the server received 34 bits, but as can be seen from the monitor this is on the beginning of a feature being sent.Figure 5-4 shows how a sent feature changes the bytes received. As can be seen, three users cause the server to receive extra bytes, but the majority of the bytes being used to insert the feature into the database using the connected FeatureInsertionService. The overall performance of the CAx Connect server shows that the features being sent require the same amount of bandwidth, which verifies equations 5.1 and 5.2.

5.2 Bandwidth Efficiency Improvements

Without the CAx Connect implementation, the bandwidth of NX Connect was consumed because of the constant check for changes to the database. As a result, an increase in the number clients connected to the database will increase the amount of bandwidth consumed. With the CAx Connect implementation, there is no consumption for a stand still connection between the client and the server, significantly reducing the amount of bandwidth consumed, especially for larger numbers of connected clients. Features being sent from NX Connect still require a certain amount of bytes to be sent, depending on the feature, to the database. When these features are created, the same amount of information must be sent to all clients whether the CAx Connect implementation is used or not. However, the clients no longer need to ask for changes, but receive the changes from CAx Connect through push technology, allowing for event driven feature creation.

A preliminary test to measure latency was also conducted utilizing resources from Brigham Young University (BYU) and Georgia Institute of Technology (GT). A case study is currently underway between the two universities conducting research utilizing the CAx Connect architecture. Multiple users simultaneously created features from workstations in GT communicating to servers in BYU utilizing the NX Connect implementation and the average time turnaround time was recorded using a stop watch timer. Using the previous NX Connect prototype without the game server, the average time from a feature created in GT to the same feature appearing on another workstation in GT was approximately 3.2 seconds. The same test was conducted using the new CAx Connect implementation and the preliminary result shows a noticeable improvement of more than ten percent reduction on the turnaround time.

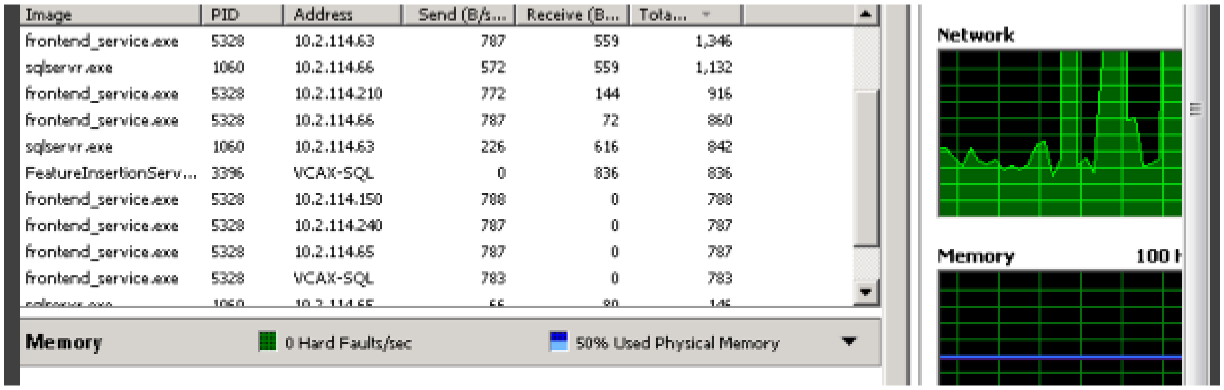


Image	PID	Address	Send (B/s...	Receive (B...	Tota...
frontend_service.exe	5328	10.2.114.63	787	559	1,346
sqlservr.exe	1060	10.2.114.66	572	559	1,132
frontend_service.exe	5328	10.2.114.210	772	144	916
frontend_service.exe	5328	10.2.114.66	787	72	860
sqlservr.exe	1060	10.2.114.63	226	616	842
FeatureInsertionServ...	3396	VCAX-SQL	0	836	836
frontend_service.exe	5328	10.2.114.150	788	0	788
frontend_service.exe	5328	10.2.114.240	787	0	787
frontend_service.exe	5328	10.2.114.65	787	0	787
frontend_service.exe	5328	VCAX-SQL	783	0	783

Memory 🟩 0 Hard Faults/sec 🟦 50% Used Physical Memory

Figure 5-4: Network tool monitoring feature sent using CAx Connect

5.3 Roadmap

The roadmap to creating a successful MMORPG architecture for multi-user collaborative tools now has a foundation. Once a tool is available to create services, those services will form a server that can be run on multiple machines. A client can be created to connect to the services front end, and the multi-user collaborative tool can interface with the client. The server will need to interface with the backend data structure to store master copies of files.

With the architecture created, the next step is to create callback functions for new features that are to be passed back and forth between the multi-user tool and its backend data structure. The server is only as good as the callback functions that it utilizes. When the desired number of callbacks is created, the integration is complete.

A queue should be integrated with the FIS in order to handle multiple message passing to the database as well, since multiple users will be logging into the multi-user tool and each will need to pass features to the database.

6 SUMMARY AND CONCLUSIONS

An in depth study of fully distributed client server architectures was performed and the results show that an integration with multi-user tools would be advantageous as this type of server provides push technology, allowing for scalability and reduced bandwidth consumption. The integration was implemented by creating a network of services that perform specific duties, including but not limited to feature propagation, message passing, and database insertion. In order to complete the integration, several requirements had to be met, including:

- Creating services for client to server, server to client, and server to server communication.
- Creating a client that can communicate with the services.
- Creating a service to database communication system
- Integrating multi-user CAx tool by providing feature support

The services were created by utilizing Nel, an open source game server development tool kit that allows the creation of services which form a cohesive MMORPG data structure for passing information across a network of servers. NX Connect was chosen as a tool to integrate, since it acts as a synchronous collaborative tool which allows multiple users to modify and create CAD features simultaneously across separate geographical regions. Features were support by specifying SOAP objects that would allow NX Open API features to be inserted, which are then serialized and passed to the MMORPG architecture created called CAx Connect, which was able to pass the serialized message to various services, using push technology. The architecture was tested and proven to have a more efficient management of bandwidth compared to the current client-server architecture that NX Connect currently utilizes. This was confirmed by pinging the network for latency and monitoring the number of bytes that it received from each client. CAx Connect also provides scalability by allowing its services to run on multiple physical machines, harnessing the processing power of each, while the current implementation of NX Connect cannot do this.

6.1 **Future Work**

While the value of the method has been successfully shown, the system is still limited by the number of services and physical machines that are available. It is also integrated into a multi-user CAD tool and it is recommended to be used in other types of multi-user engineering tools. The benefits of expanding on the limitations and of the integration with another type of CAx tool would further show the benefit of choosing this type of architecture.

One way of removing the service limitation is to create new services that provide extra functionality to the multi-user tool. The amount of new functionality depends on the creativity of the user. For example, a position tracking service could be created so that the location of other users in the multi-user tool can be known, and viewports can be swapped with the click of a button.

Latency could also be measured programmatically using a tick timer to track when features are sent and when they are received so that information can be more accurate on how well this implementation works with larger number of users over long distances, as this tool can be utilized for global collaboration.

Adding security to the TCP/IP protocols would also be something to consider for perpetuating the use of this architecture into industry. Security protocols currently already exist for secure TCP/IP encryption such as SSL, OpenSSL, or GnuTLS. Since industry requires some verification of the user identity and there would be multiple users connecting to this architecture, user identification and hack prevention would need to be implemented to secure the information contained in the CAx files to maintain data integrity. Since this architecture is layered similarly to all networks, there is a specific layer that handles TCP/IP communication and encryption can be added to the files containing the sockets found in code/nel/src/net/tcp_sock.cpp and code/nel/include/nel/net/tcp_sock.h.

With further work, services can also be used to spread computation across multiple processors. Adding more physical machines is no problem for the services, as they can run on multiple machines. This would remove the other limitation. When integrated with a different type of application, like a finite element application, the computation could be broken into components and spread over an entire network of physical server boxes for computation, significantly decreasing the amount of time to perform the computation and increasing the benefit of integrating this type of architecture. This could also be applicable for not only finite element applications, but also computational fluid dynamics (CFD) applications, heat transfer applications, thermodynamic analysis, multi-user assemblies and more. The research of this method would only stand to benefit more by expanding upon this architecture.

REFERENCES

Assiotis, M., and Velin, T. "A Distributed Architecture for Massive Multiplayer Online Role Playing Games," Massachusetts Institute of Technology, 13 May 2005. Web. 20 Dec 2010.

Bidarra, R., Van Den Berg, E., and Bronsvoort, W. F., ‘‘Interactive Facilities for Collaborative Feature Modeling on the Web,’’ *Proc. of the Tenth Portuguese Conference on Computer Graphics*, Lisbon, Portugal, pp. 43–52, 2001.

Caltagirone, S., Keys, M., Schlief, B., and Willshire, M., PhD, “Architecture For Massively Multiplayer Online Role Playing Game Engine,” *Consortium for Computing Sciences in College,* Web. 18 Jan 2012.

CCP Games. “This Weekend: The Alliance Tournamet Finals.” Weblog entry, *EveOnline Insider*. 17 June 2010. 9 September 2011. http://www.eveonline.com/devblog.asp?a=blog&bid=771

Cera, C. D., Regli, W., Braude, I., Shapirstein, Y., and Foster, C.,"A Collaborative 3D Environment for Authoring of Design Semantics," *IEEE,* 2002. Web. 20 Dec 2010.

Chan, S., Wong, M., and Ng, V., 1999, ‘‘Collaborative Solid Modeling on the WWW,’’ *Proc. of the 1999 14th ACM Symposium on Applied Computing*, San Antonio, Texas, pp. 598–602.

Chen, H. and Tien, H. “Application of Peer-to-Peer Network for Real-Time Online Collaborative Computer-Aided Design.” *Journal of Computing in Civil Engineering*, 21, 2. 22 June 2006.

Chu, H. 8 Sept 2008. “Building a simple yet powerful MMO game architecture, Part 1-3.” www.ibm.com/developerworks. Retrieved 9 Sept 2011 from http://www.ibm.com/developerworks/library/ar-powerup1/

Fan, L. Q., A. Senthil Kumar, B. N. Jagdish, and S. H. Bok. "Development of a Distributed Collaborative Design Framework within Peer-to-peer Environment," *Computer-Aided Design* 40.9 (2008). ACM Digital Library. Sept 2008. Web. 22 June 2010. http://portal.acm.org/citation.cfm?id=1413004 .

Fuh, J.Y.H. and Li, W.D. “Advances in collaborative CAD: the-state-of-the-art,” *Computer-Aided Design*, 37, 5, pp. 571-581. 15 April 2005. Web. 2 Sept 2011.

Foster, C. V., Shapirstein, Y., Cera, C. D., and Regli, W. C., ‘‘Multi-User Modeling of Nurbs-Based Objects,’’ *Proceedings of DETC’01, 2001 ASME Design Engineering Technical Conferences & Computers and Information in Engineering Conference* ~DETC2001/CIE-21256!, Pittsburgh, Pennsylvania. Web. 22 June 2011. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=999787&userType=&tag=1

GauthierDickey, C., Zappala, D., and Lo, V., “A Fully Distributed Architecture for Massively Multiplayer Online Games,” *SIGCOMM ’04 Workshops*, Sept 3, 2004.

"History of Massively Multiplayer Online Games." *Wikipedia, the Free Encyclopedia*. 19 Aug 2007. Web. 13 Feb 2012. http://en.wikipedia.org/wiki/History_of_massively_multiplayer_online_games.

Kao, Y. C., and Lin, G. C. I., 1998, ‘‘Development of a Collaborative CAD/CAM System,” *Pergamon Robotics and Computer-integrated Manufacturing 14* (1998), 55-68. 13 May 1997. Web. 21 Dec 2011.

Lang, J., 2011. Service Architecture, dev.ryzom.com, Retrieved 9 Sept 2011 from http://dev.ryzom.com/projects/ryzom/wiki/ServiceArchitecture

Lee, J. Y., 2001, ‘‘Shape Representation and Interoperability for Virtual Prototyping in a Distributed Design Environment,’’ *Int. J. Adv. Manuf. Techno*, 17, pp. 425–434.

Min, L., Shuming, G., “Real-Time Collaborative Design With Heterogeneous CAD Systems Based on Neutral Modeling Commands,” *Journal of Computing and Information Science in Engineering*, 7, 113. June 2007. Web. 27 Feb 2012.

Orland, K. (2008-05-20). "GFH: The Real Life Lessons Of WoW's Corrupted Blood". Gamasutra. United States: *Game Developer*.

PC Zone Staff. “Looking Back…World of Warcraft.” Computerandvideogames.com (CVG). 4 Jan 2005. Web.12 Sept 2011. http://www.computerandvideogames.com/131791/features/looking-back-world-of-warcraft/

Ramani, K., Agrawal, A., Babu, M., and Hoffman, C. "CADDAC: Multi-Client Collaborative Shape Design System with Server-based Geometry Kernel." *J. Comput. Inf. Sci. Eng* 3.2 (2003): 170-74. June 2003. Web. 20 Dec 2010. http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=JCISB6000003000002000170000001&idtype=cvips&gifs=yes

Ryskamp, J. “Developing Intelligent Engineering Collaboration Tools Through the use of Design Rationale.” MS Thesis. Brigham Young University, Utah, 2010. Web 9 Sept 2011.

Ryzom Development. Jean-Phillipe Lang. 2006. Redmine. 21 Jan. 2012. http://dev.ryzom.com.

Shu, L., and Flowers, W., 1992, ‘‘Groupware Experience in Three-Dimensional Computer-Aided Design,’’ *Proceedings of CSCW’92*, pp. 179–186.

Zheng, Y., Shen, H., & Sun, C., 2008. “Adapting single-user autoCAD system to support realtime collaborative design: issues, challenges and achievements.” *Proceedings of the Tenth International Workshop on Collaborative Editing Systems in conjunction with ACM Conference on Computer Supported Cooperative Work.*

APPENDIX A. INSTALLATION AND EXECUTION OF NEL

All files for CAx Connect were built using CMake from the source code found at dev.ryzom.com, as specified in Chapter 4 section 1. The resulting construction shown in Figure A-1 demonstrates where CAx Connect needs to be installed on the C: Drive. In the code folder is the source code for NEL, the services folder is the services created for this research, and the build folder contains the Ryzom.sln file which is the solution containing all of the projects to build services for NEL.

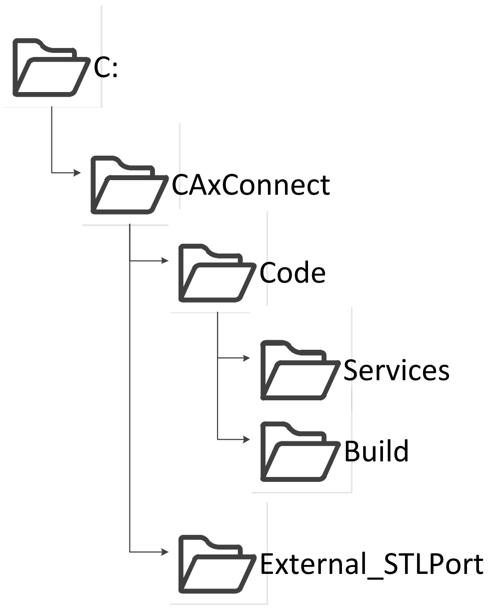


Figure A-1: File tree

This server utilized CMake to build, so configuration of the server can be changed, however, without changing the configuration, the files must be saved in the C: Drive as shown in Figure A-1.

The source code is available on Tortoise SVN at inside the MMORPG folder. The address is: svn://it.et.byu.edu:1667/svn/MMORPG/CAxConnect which contains the structure shown in Figure A-1. In order to ensure a successful build, Microsoft SDK 7.1 must be installed as well as the June 2010 or later DirectX SDK. When the code is downloaded, it can be run by building the Ryzom.sln solution found in the build folder.

The Front End Service (FrontEndService.cfg) files looks like the following:

```
NSHost = "localhost";

WindowStyle = "WIN";

AcceptInvalidCookie = 1;

NegFiltersDebug = { "NET" };

DontUseAES = 1;
```

Where NSHost is the IP address of the computer running the naming service, windows style sets the type of window that the FS will display, and DontUseAES =0 means to use the AES, which is explained in Appendix B.

The feature administration service (feature_administration_service.cfg) looks like the following:

```
NSHost = "localhost";
WindowStyle = "WIN";
DontUseAES = 1;
```

Where NSHost is the IP address of the computer with the naming service.

To run the naming service, two configuration files are needed, common.cfg and naming_service.cfg. There files look like this:

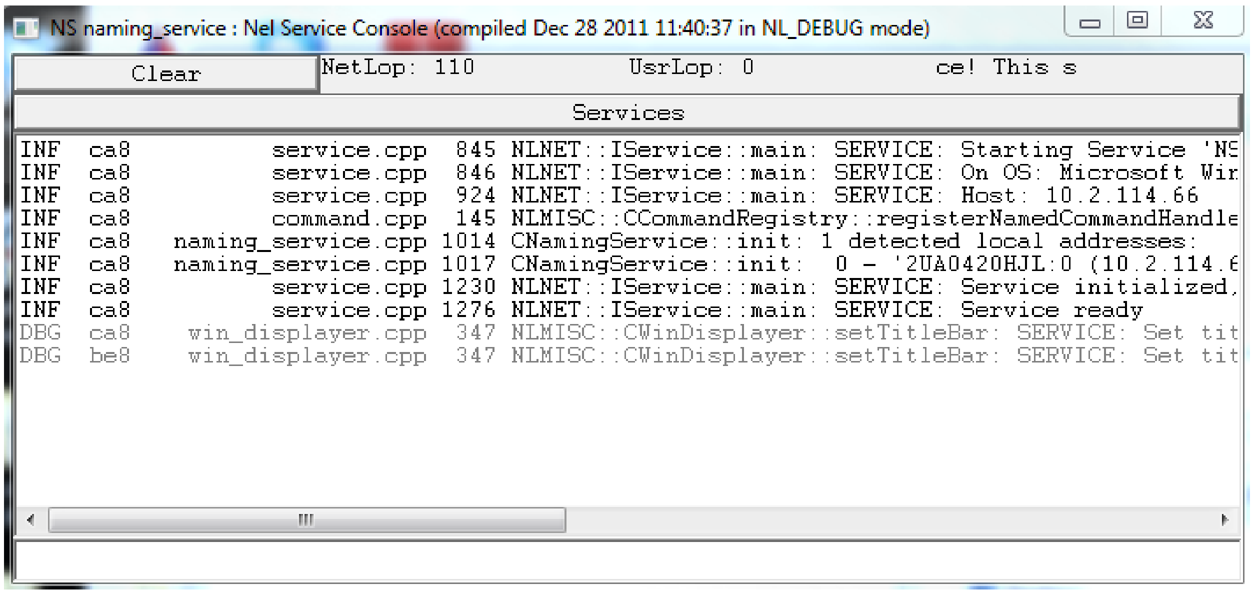
Common.cfg:

```
        // by default, use WIN displayer
        WindowStyle = "WIN";

        // by default, use localhost to find the naming service
        NSHost = "localhost";

        NegFiltersDebug    += { "NETL" };
        NegFiltersInfo     += { "NETL" };
NegFiltersWarning  += { };
```

Naming_service.cfg:

```
        // link the common configuration file
        RootConfigFilename = "common.cfg";

        DisplayedVariables += { "", "@Services|nsServices" };

        SId = 1;

        DontUseNS = 1;
        DontUseAES = 1;

        NegFiltersDebug = { "NETL" };
        NegFiltersInfo = { "NETL" };

        UniqueOnShardServices = { "WS" };
UniqueByMachineServices = {};
```

To run the services for CAxConnect outside of debug mode, similar configuration files need to be included in the same directory as the executables that are generated from the projects. The executables for the project can be found in the build/bin/Debug folder.



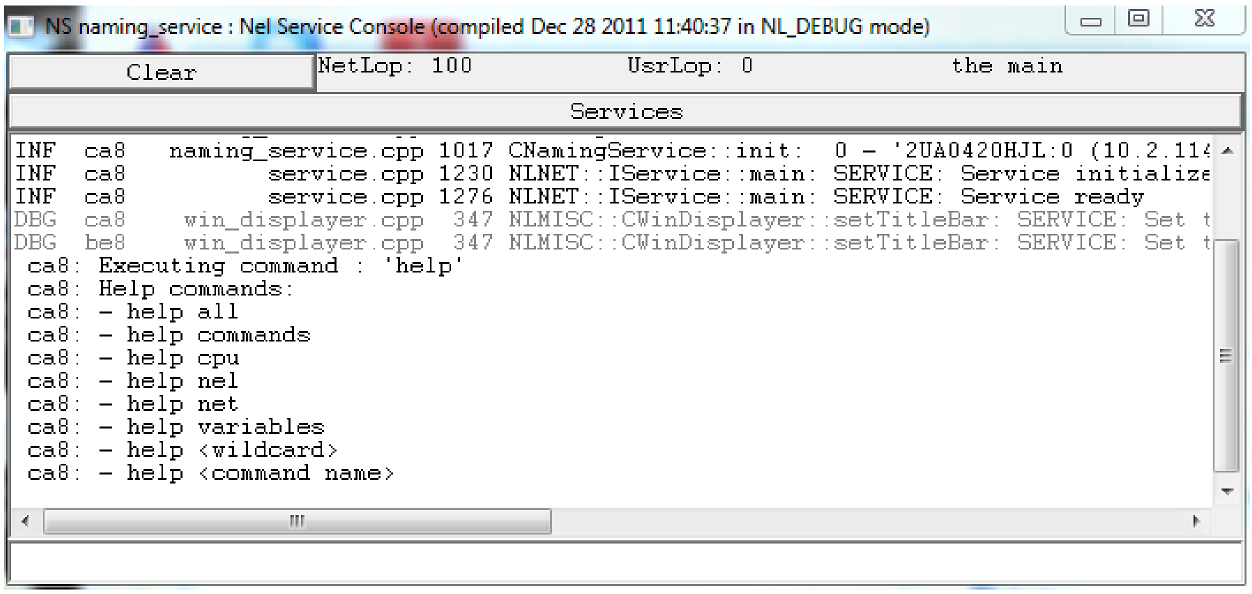
NS naming_service : Nel Service Console (compiled Dec 28 2011 11:40:37 in NL_DEBUG mode)
Clear
Services

```
NetLop: 110            UsrLop: 0              ce! This s
```

```
INF  ca8          service.cpp  845 NLNET::IService::main: SERVICE: Starting Service 'NS
INF  ca8          service.cpp  846 NLNET::IService::main: SERVICE: On OS: Microsoft Wir
INF  ca8          service.cpp  924 NLNET::IService::main: SERVICE: Host: 10.2.114.66
INF  ca8          command.cpp  145 NLMISC::CCommandRegistry::registerNamedCommandHandle
INF  ca8   naming_service.cpp 1014 CNamingService::init: 1 detected local addresses:
INF  ca8   naming_service.cpp 1017 CNamingService::init:  0 - '2UA0420HJL:0 (10.2.114.6
INF  ca8          service.cpp 1230 NLNET::IService::main: SERVICE: Service initialized,
INF  ca8          service.cpp 1276 NLNET::IService::main: SERVICE: Service ready
DBG  ca8    win_displayer.cpp  347 NLMISC::CWinDisplayer::setTitleBar: SERVICE: Set tit
DBG  be8    win_displayer.cpp  347 NLMISC::CWinDisplayer::setTitleBar: SERVICE: Set tit
```

Figure A-2: Nel service window

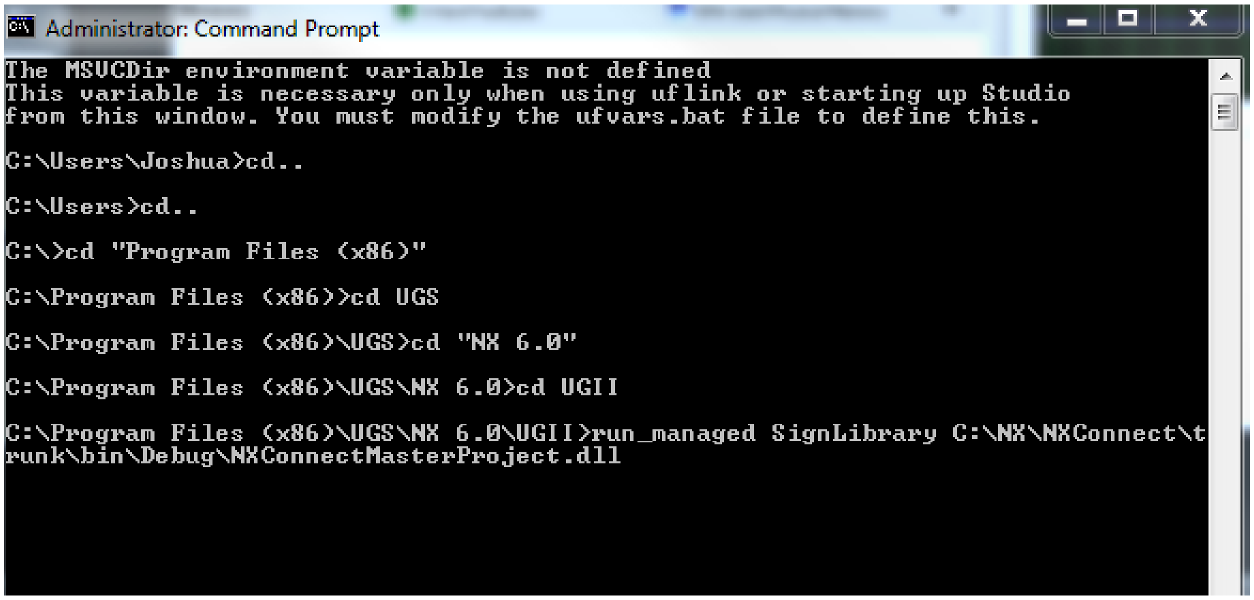
When the server is running the executables for the various services, it will display that the service is connected in the various windows that are generated. Any features that are passed are programmed inside of the features to display the information being passed as shown in Figure A-2. By typing in "help" into the window, you can access all of the NEL commands to monitor the service (Figure A-3).


NS naming_service : Nel Service Console (compiled Dec 28 2011 11:40:37 in NL_DEBUG mode)
Clear
NetLop: 100
UsrLop: 0
the main
Services

```
INF  ca8   naming_service.cpp 1017 CNamingService::init:  0 - '2UA0420HJL:0 (10.2.114
INF  ca8          service.cpp 1230 NLNET::IService::main: SERVICE: Service initialize
INF  ca8          service.cpp 1276 NLNET::IService::main: SERVICE: Service ready
DBG  ca8    win_displayer.cpp  347 NLMISC::CWinDisplayer::setTitleBar: SERVICE: Set t
DBG  be8    win_displayer.cpp  347 NLMISC::CWinDisplayer::setTitleBar: SERVICE: Set t
 ca8: Executing command : 'help'
 ca8: Help commands:
 ca8: - help all
 ca8: - help commands
 ca8: - help cpu
 ca8: - help nel
 ca8: - help net
 ca8: - help variables
 ca8: - help <wildcard>
 ca8: - help <command name>
```

Figure A-3: Nel service help menu

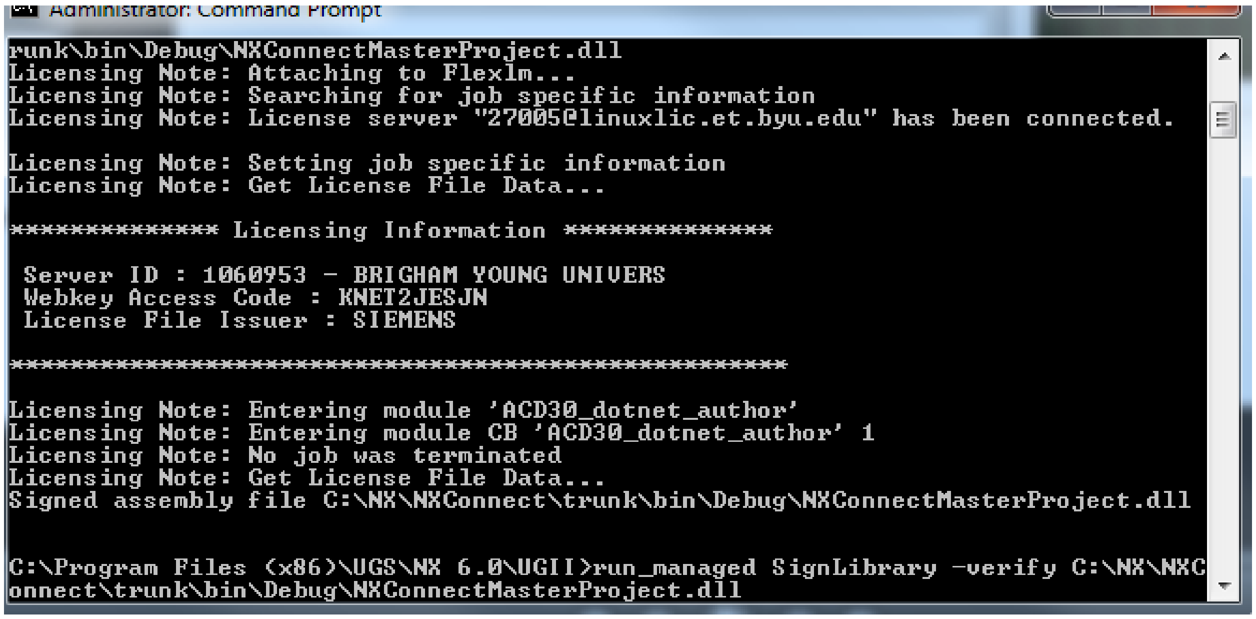
As messages are propagated and clients connect, the display window will show features being passed as well as how many clients are connected in the corresponding window that the service or client is connecting to (i.e. the client connects to the FS, so the FS window will show client to FS interactions).



```
The MSVCDir environment variable is not defined
This variable is necessary only when using uflink or starting up Studio
from this window. You must modify the ufvars.bat file to define this.

C:\Users\Joshua>cd..

C:\Users>cd..

C:\>cd "Program Files (x86)"

C:\Program Files (x86)>cd UGS

C:\Program Files (x86)\UGS>cd "NX 6.0"

C:\Program Files (x86)\UGS\NX 6.0>cd UGII

C:\Program Files (x86)\UGS\NX 6.0\UGII>run_managed SignLibrary C:\NX\NXConnect\t
runk\bin\Debug\NXConnectMasterProject.dll
```

Figure A-4: Signature application for NX Connect

In order to connect NX Connect to the tool, the client.cfg file and the caxclient.dll must be put into the same directory as the unigraphics NX executable file (ugraf.exe) which is located in the UGS/NX6.0/UGII folder. The NXConnectMasterProject.dll file which runs inside of NX after pressing ctrl+u will run NX Connect. The NXConnectMasterProject.dll can be placed anywhere on the computer. However, to run this .DLL file outside of the network, a signature must be applied to the .DLL so that other users will not require a dotnet author license. Siemens



```
Administrator: Command Prompt
runk\bin\Debug\NXConnectMasterProject.dll
Licensing Note: Attaching to Flexlm...
Licensing Note: Searching for job specific information
Licensing Note: License server "27005@linuxlic.et.byu.edu" has been connected.

Licensing Note: Setting job specific information
Licensing Note: Get License File Data...

************** Licensing Information **************

 Server ID : 1060953 - BRIGHAM YOUNG UNIVERS
 Webkey Access Code : KNET2JESJN
 License File Issuer : SIEMENS

****************************************************

Licensing Note: Entering module 'ACD30_dotnet_author'
Licensing Note: Entering module CB 'ACD30_dotnet_author' 1
Licensing Note: No job was terminated
Licensing Note: Get License File Data...
Signed assembly file C:\NX\NXConnect\trunk\bin\Debug\NXConnectMasterProject.dll


C:\Program Files (x86)\UGS\NX 6.0\UGII>run_managed SignLibrary -verify C:\NX\NXC
onnect\trunk\bin\Debug\NXConnectMasterProject.dll
```

Figure A-5: Verification of signature

provided a custom signature executable for this project which allows a 32 bit NX .DLL to run on a 64 bit Windows 7 operating system. This .DLL can be found in svn://it.et.byu.edu:1667/svn/NXConnect/run_managed. To run it, enter the same code as seen inFigure A-4. To verify that the signature is a success, similarly the SignLibrary executable has a verification command as shown in Figure A-5.

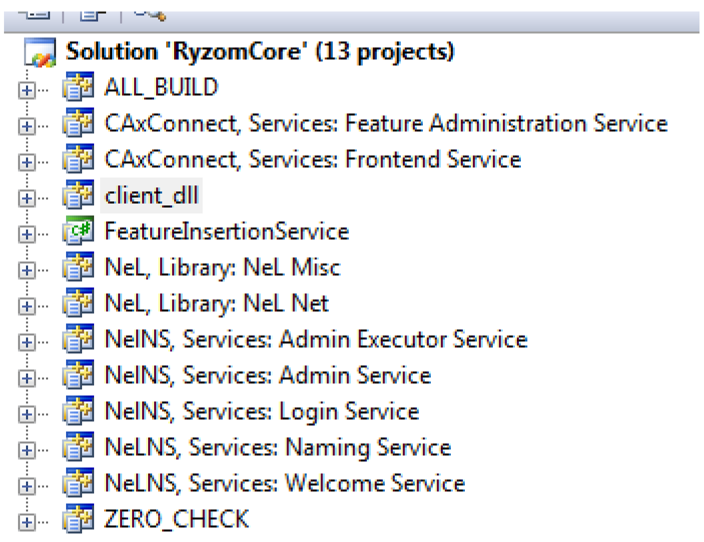
In order to run the server, the following binaries and configuration files are needed:

- FrontEndService.exe
- FeatureInsertionService.exe
- NamingService.exe
- FeatureAdministrationService.exe
- Common.cfg

- Frontendservice.cfg
- Client.cfg
- Namingservice.cfg
- Featureadministrationservice.cfg

APPENDIX B. SERVICES DEFINITIONS

Several of the services (Figure B-1) are unnecessary to run CAx Connect but will be useful for



```
Solution 'RyzomCore' (13 projects)
  ALL_BUILD
  CAxConnect, Services: Feature Administration Service
  CAxConnect, Services: Frontend Service
  client_dll
  FeatureInsertionService
  NeL, Library: NeL Misc
  NeL, Library: NeL Net
  NeINS, Services: Admin Executor Service
  NeINS, Services: Admin Service
  NeINS, Services: Login Service
  NeLNS, Services: Naming Service
  NeLNS, Services: Welcome Service
  ZERO_CHECK
```

Figure B-1:Service source code

user development. A description of each service is given below:

- Naming Service (NS): Acts as a directory for other services, tracking IP address and dynamically assigning IP addresses to services that start up. Start this executable before all services.
- Frontend Service (FS): All clients connect to this service. It acts as a relay from the server to all clients. The executable for this project must be started before any clients log on. This service does not get an IP address assigned dynamically, as the clients must specify in the configuration file which FS it should connect to.

- Feature Administration Service (FAS): All messages passed to this service from the FS are propagated to all running FS to get information to all clients. Registers with NS to get an IP address assigned dynamically.
- client_dll: project contains the source code to modify the Game Client DLL mentioned in the research which handles message passing between CAx Connect and NX Connect.
- FeatureInsertionService: Executable that connects to the FS (ideally this should connect to the FAS) to receive messages. Deserializes the messages and inserts the features contained in the messages into the database. Sends messages via Game Client DLL to FS if there are errors with insertion.
- NEL MISC: Contains source code for NEL API calls for all core types, functionality and utilities necessary to use NET (dev.ryzom.com or www.opennel.org).
- NEL NET: Contains source code for NEL NET API calls which handles client and server code communication.
- NELNS Admin Executor Service (AES) (not used): If NEL server is connected to a web service with the database located on it, this allows HTTP language construction of a service which can dynamically change the client server connection to any clients, allowing runtime load balancing of clients on FS.
- NELNS Admin Service (AS) (not used): Provides admin access to the AES.
- NELNS Login Service (LS) (not used): Provides logging services.

  NELNS Welcome Service (WS) (not used): Provides additional logging services in conjunction with the LS.

APPENDIX C. CLIENT CONFIGURATION

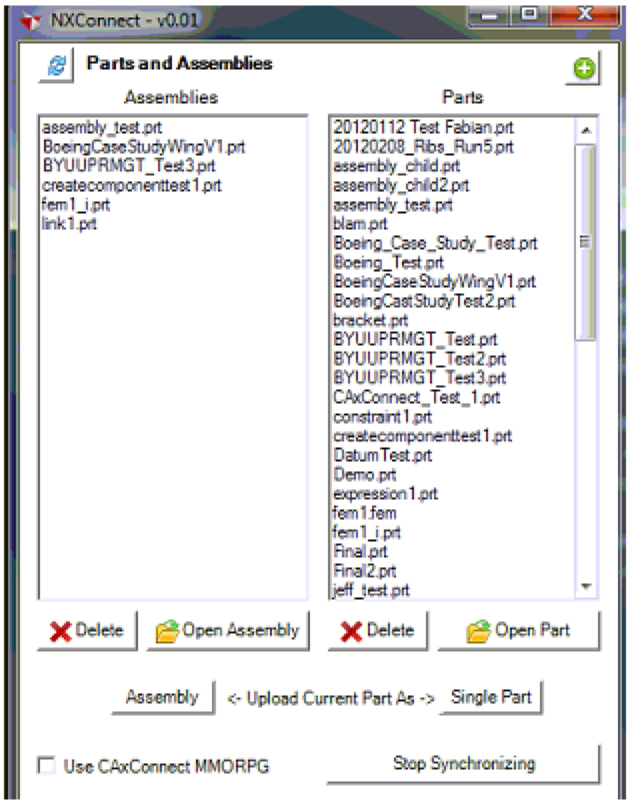
In order for NX Connect or any multi-user CAx tool to connect to CAx Connect, a configuration file as well as the caxclient.dll file must be placed inside of the same folder that the CAx executable is located in. For example, if NX 6.0 32 bit is utilized for NX Connect, the client.cfg and caxclient.dll files must be placed in C:\Program Files (x86)/UGS/NX6.0/UGII. In order for the multi-user tool to connect to CAx Connect, the caxclient.dll file needs to have its initialize function called, which searches for the IP address of the machine that CAxConnect front end services are running on. For the client_dll, the configuration file (client.cfg) looks like the following:

```
LSHost = "localhost";
```

Where LSHost is the IP address of the computer that the Front End Service (FS) is running on. If the client is located on a server with a port number, the port number must be attached to the address in the config file. This would then become, for example:

```
LSHost = “10.2.114.66:37000”;
```

To run the client, NX must be run using the ugraf.exe file. When NX has compiled and is running on the client, a variety of executables, .dll, jar, or class files can be run by pressing ctrl+u. The NXConnectMasterProject.dll file should be selected to run the NX Connect plugin. Inside of the master form window, there is an option to select USE CAxConnect MMORPG (Figure C-1).



assembly_test.prt
BoeingCaseStudyWingV1.prt
BYUUPRMGT_Test3.prt
createcomponenttest1.prt
fem1_i.prt
link1.prt

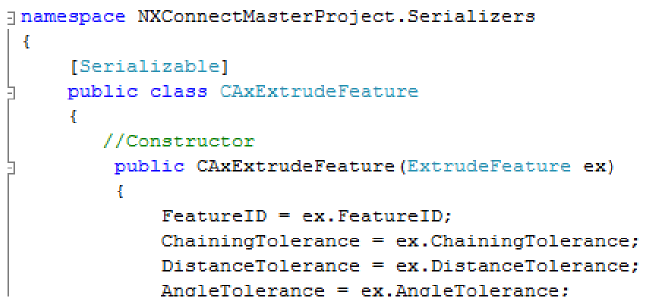
20120112 Test Fabian.prt
20120208_Ribs_Run5.prt
assembly_child.prt
assembly_child2.prt
assembly_test.prt
blam.prt
Boeing_Case_Study_Test.prt
Boeing_Test.prt
BoeingCaseStudyWingV1.prt
BoeingCastStudyTest2.prt
bracket.prt
BYUUPRMGT_Test.prt
BYUUPRMGT_Test2.prt
BYUUPRMGT_Test3.prt
CAxConnect_Test_1.prt
constraint1.prt
createcomponenttest1.prt
DatumTest.prt
Demo.prt
expression1.prt
fem1.fem
fem1_i.prt
Final.prt
Final2.prt
jeff_test.prt

Figure C-1: NX Connect master form

When this option is selected, the configuration of NX Connect will no longer propagate message through the data sync module but will send messages to CAx Connect.

APPENDIX D. LINQ OBJECTS

Objects generated for each feature within NX Connect were created using LINQ, which creates objects that can be used to insert into Microsoft SQL Server database. However, LINQ objects cannot be serialized into a formatter and in order to pass features from a multi-user CAx tool, CAx Connect requires that features be packaged into strings so pass through the server. Serializable objects are created inside the multi-user tool can be utilized by utilizing the .DLL file created inside of the database insertion service written in C#. Figure D-1 shows a code snippet of the extrude serializable class which contains all of the same feature information used to generate a LINQ object, which also mirrors the code of the NX Open API.

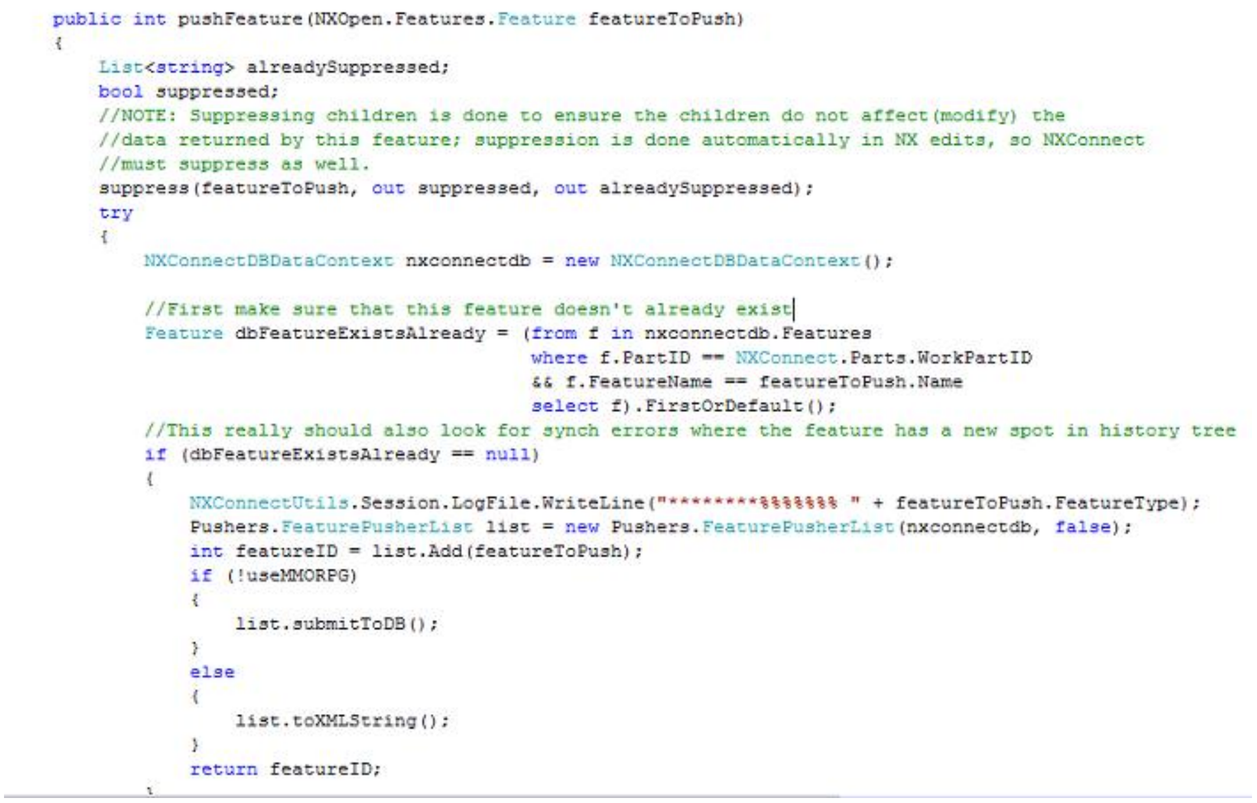


```
namespace NXConnectMasterProject.Serializers
{
    [Serializable]
    public class CAxExtrudeFeature
    {
        //Constructor
        public CAxExtrudeFeature(ExtrudeFeature ex)
        {
            FeatureID = ex.FeatureID;
            ChainingTolerance = ex.ChainingTolerance;
            DistanceTolerance = ex.DistanceTolerance;
            AngleTolerance = ex.AngleTolerance;
```

Figure D-1: Serializer source code

Features that are created from within NX Connect are found by parsing the undo marks found within NX. When the undo mark for a given feature is found, the data capture module (DataCaptureModule.cs) sends a message to create a pusher. All features that are supported within NX Connect have an associated feature pusher that matches the NX Open API. If a change has occurred by the creation of an undo mark, the NX Connect Controller

(NXConnectController.cs) is called, which then puts all of the features that were created within NX into a pusher list and submits to the database. However, when CAx Connect is initialized, the features from the undo mark are not submitted to the database directly, rather, they are serialized into a SOAP object, then put into a string, and are sent to the CAx Connect server. This code is shown in Figure D-2 and is also found within the NXConnectController.cs file. If the Use MMORPG option is selected in the NX Connect Master Form, then the feature will call toXMLString() on the pusher list and will send the data to CAx Connect.



```
public int pushFeature(NXOpen.Features.Feature featureToPush)
{
    List<string> alreadySuppressed;
    bool suppressed;
    //NOTE: Suppressing children is done to ensure the children do not affect(modify) the
    //data returned by this feature; suppression is done automatically in NX edits, so NXConnect
    //must suppress as well.
    suppress(featureToPush, out suppressed, out alreadySuppressed);
    try
    {
        NXConnectDBDataContext nxconnectdb = new NXConnectDBDataContext();

        //First make sure that this feature doesn't already exist
        Feature dbFeatureExistsAlready = (from f in nxconnectdb.Features
                                          where f.PartID == NXConnect.Parts.WorkPartID
                                          && f.FeatureName == featureToPush.Name
                                          select f).FirstOrDefault();
        //This really should also look for synch errors where the feature has a new spot in history tree
        if (dbFeatureExistsAlready == null)
        {
            NXConnectUtils.Session.LogFile.WriteLine("********%%%%%%% " + featureToPush.FeatureType);
            Pushers.FeaturePusherList list = new Pushers.FeaturePusherList(nxconnectdb, false);
            int featureID = list.Add(featureToPush);
            if (!useMMORPG)
            {
                list.submitToDB();
            }
            else
            {
                list.toXMLString();
            }
            return featureID;
```

Figure D-2: Push feature source code within NX Connect controller