2010-11-30

A Flexible Context Architecture for a Multi-User GUI

Yue Xu
Brigham Young University - Provo

A Flexible Context Architecture for a Multi-User GUI

Yue Xu

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

Master of Science

Department of Mechanical Engineering

Brigham Young University

December 2010

ABSTRACT

A Flexible Context Architecture for Multi-User GUI

Yue Xu

Department of Mechanical Engineering

Master of Science

This thesis focuses on the design and development of a flexible context architecture for a multi-user GUI that serves several users engaged in collaborative computer-aided applications (CAx). The objective of this thesis is to extend previous research into multi-user GUI's by providing multiple users with a flexible context interface for interaction with other users working on the same part at distributed locations. The investigation will consider how distributed users, through user interfaces, interact to simultaneously build models and how interaction context might be presented to regulate the way they wish to interact.

The implementation integrates a Multi-User GUI (MUG) with NX Connect, a multi-user CAD prototype for Siemens NX. NX MUG uses agent software to render the user prototype outside of NX and NX Connect. This generalizes the interface so that it could be used with other engineering applications where several users wish to collaborate. The Multi-user GUI enables users to view a collaborating user's workspace, send/receive messages between multi-users, and is capable of translating text interactions in different languages, while skyping with other users. This research will have a profound impact on collaborative teams in reducing barriers to effective communication. This research will also enhance the existing NX Connect multi-user prototype by providing collaborative interaction support among the multi-users.

Keywords: Yue Xu, multi-user CAx, multi-user GUI, collaborative design

ACKNOWLEDGMENTS

I express my appreciation for all of those who helped me directly on this thesis. I want to express thanks to my Committee Chair, Dr. Edward Red, for giving me critical guidance and motivation to explore the multi-user GUI. I also thank my advisor Dr. Greg Jensen for his support and encouragement and for the opportunities to work in the ParaCAD research group. I would also like to thank the other students in the ParaCAD research group at BYU for all the in-put and insight I have received during my research.

Thanks to the Department of Mechanical Engineering for assistance during these years. Special thanks go to Miriam Busch for her kindness and support.

Lastly, I am particularly grateful for my parents for their support and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Computer support for cooperative work (CSCW) is the computer technology to support collaborative activities and requires “the development of models and tools to support the construction of these distributed, multi-user interfaces and the development of associated software architectures” [26]. It also requires the construction of applications which exploit multi-user interfaces to promote their collaborative activities in a multi-user environment. There is the need to achieve co-development of parts by designers who are spread out among a number of different locations and “the associated interfaces run across a number of workstations” in a distributed environment [26]. This need to support multi-user interface capability has resulted in “a merging of the concerns of user interface software and distributed systems” [26].

This thesis will investigate a flexible context architecture for a multi-user graphical user interface (MUG) that can be used in collaborative engineering activities. The aim of the architecture is to provide collaborative interface flexibilities for user interface designs.

1.1 Problem Statement

Today, product development practices assign the steps of design, analysis, and process planning to specific departments and ultimately to responsible individuals, all within a highly controlled serial process [12]. This serialized design process is largely driven by a set of single user core computer-aided (CAx) applications.

The architecture today “may refer to the activity of designing any kind of system and is often used in the IT world” [25]. Single user architectures, including the user interfaces that characterize both commercial computer-aided application software and computer operating systems, greatly inhibit collaborative engineering, in spite of the numerous research efforts into product team cooperation. A flexible context allows “the activities of one user to be reflected on other users screens and is supported by sharing application information” [26]. Bentley et al. [26] state that sharing is “the principle means of promoting cooperation, and the real-time presentation and manipulation of shared information is the main function of cooperative, multi-user interfaces”.

The objective of this thesis is to build flexible context architectures into the multi-user GUI that serves several users engaged in collaborative CAx. We have already developed a multi-user client-server collaborative software prototype at BYU called NX Connect We used the API (Application Programming Interface) set available in Siemens NX to extend the single user commercial CAx software to a multi-user environment where several users can simultaneously construct and edit the same model, but without a formal multi-user GUI to provide a flexible context for the collaborative activities. This thesis will extend previous research into multi-user GUI’s by providing the multi-user with a flexible context interface for interaction with other users working on the same part at distributed locations. The investigation will consider how distributed users, through user interfaces, interact to simultaneously build models, and how interaction context might be presented to regulate the way they wish to interact.

1.2 Thesis Objectives

This thesis has the following objectives:

1. Investigate and develop architectures for multi-user interfacing of localized and distributed collaborators applying CAx applications, including agent software that can be configured for user context preference.
2. Determine how to integrate filters to provide personalized and selective contextual presentation of collaborating user interactions. A filter provides a function/algorithm to limit collaborating data input, to provide a different view of that data, or to promote multi-user awareness. Filters could be mapped to Multi-User GUI (MUG) menu items or selection icons, etc.
3. Develop MUG prototypes that are consistent in look and feel with current commercial CAx GUI's, but can be invoked or hidden. The intent is to develop software libraries that can be linked to CAx API libraries, and then eventually become a member of the API library after commercialization by a CAx company. In addition, a prioritization method will be considered that allows manager oversight of multi-user sessions.
4. Demonstrate that MUG's can be practically used to enhance multi-user CAx; and that product personnel of different cultures can navigate CAx collaboration effectively, in spite of cultural differences.

1.3 Delimitation of the Problem

This purpose of this research is to investigate and propose a suitable architecture for a MUG. Because the current API's for commercial CAx have been built assuming single user architectures, a complete MUG prototype cannot be built upon the current API's due to existing API limitations. Nevertheless, a multi-user interface will be built to test and compare the NX

API functionality to the desired MUG functionality by using available interface tools and software libraries.

1.4 Anticipated Benefits

Agent software and appropriate context filters will be designed and integrated into the MUG as possible using the currently available tools. The prototype that is built will expose weaknesses of current CAx API's for developing multi-user interfaces. The investigation will expose the challenges of integrating personalized and selective contextual presentation into a CAx multi-user GUI. A related paper(s) will be published in a technical journal or technical conference and possibly in a technical periodical.

In the long run, the context architectures for multi-user interfacing of localized and distributed collaborators will be more flexible and broadly applied to CAx applications. The methods and algorithm developed in my research will assist researchers to develop robust multi-user interfaces.

2 LITERATURE REVIEW

This section has been classified into six categories: 1) multi-user GUI (MUG); 2) ideal-user interface; 3) client – server application; 4) existing MUG architectures and infrastructures 5) multi-user awareness and agent software and 6) motivation and challenges.

2.1 Multi-User Interface

The notion that two minds are better than one or two workers can get the job done in half the time suggests that true collaborative virtual design environments could be a great asset to speeding up or improving product design. In a global collaborative design environment there is the need to co-develop parts by designers at different geographical locations. When considering global manufacturing, the geographical separation makes it difficult to frequently gather all of the involved parties and departments for a meeting [16]. Sosa notes that even some engineers who sit in the same room and work on different parts of the same model create conflicts when they try to assemble the parts together [17]. Macfarlane suggests that if engineers start a complicated model on the same screen, they may avoid "no match" situations before assembling the parts [18]. But the multi-user environment challenges system capacity and reveals problems like process collisions [19] when several people are shown in the same screen. The objective of this research is to build a flexible context architecture for multi-user GUI's that can be use in collaborative CAx. This research will investigate how the graphical multi-user interface can be constructed to regulate the way distributed users interact to simultaneously build models.

One example of a multi-user interface is CoMaya [20]. CoMaya converts single-user Maya into a real-time collaboration with Autodesk and Verse. This creates a network protocol that makes multiple applications run together. For example, CoMaya allows two or more users to collaborate simultaneously to make changes to a single file. The session maintains the original user interface and functionalities of Maya, but adds the CoMaya user interface as well. User A's actions immediately appear on User B's workspace in real-time. CoMaya uses target object high-lighting and semantic sound effects [20] to indicate this concurrent work. Either of these effects can be applied to the same object and attribute, or to the same object with different attributes. With fast local response, users can modify a part of the model, or navigate on different regions of the model, from different perspectives.

2.2 Ideal User-Interface

An ideal user-interface (UI) will help a target group easily access the application and re-duce the support cost. One UI design principle is consistency, which enables the users to "build an accurate mental mode of the way it works" and require less training and support costs [21]. Similarly, most collaborative design companies try to retain the single user interface and add multi-user view windows or view sessions.

An ideal UI will deliver the messages effectively. Equivalently, an ideal multi-user UI will promote cooperation and multi-user interaction. It will provide users with encouragement and help collaboration take place with one user's awareness of others' activities. Most existing multi-user applications produce the illusion that the users are the only ones in the system. This illusion makes the users unaware of the presence of others and prohibits collaboration. A good way to overcome the illusion and enhance multi-user awareness is to establish and maintain a common context. This context is communications between users of different cultures and personalities, and “allows the activities of one user to be reflected on other users’ screens that are supported by sharing application information” [26].

2.3 Client – Server Application

The Client-Server (CS) architecture, also known as a centralized architecture, is where “a central server program handles all user input events and display output events which are routed by way of local client programs” [26]. This architecture simplifies accessing management and data consistency. Accordingly, the great advantage of the CS approach is simplicity. But this architecture does not support end-user tailoring, and it is vulnerable to “failure of the central node (server), and delayed feedback as all input and output events must travel over a network” [26].

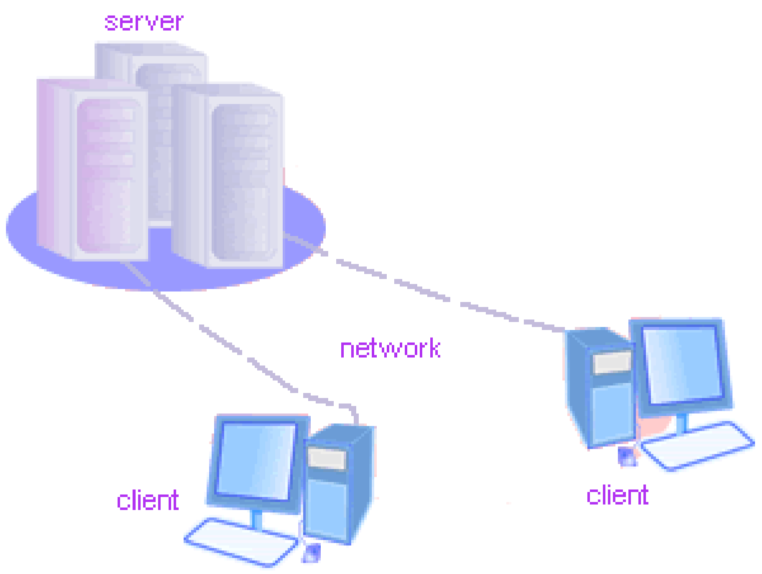


Figure 2-1: Client-Server Architecture

The literature considers two types of CS architectures that enable co-design: a thin server with strong clients and a strong server with thin clients. In the former one the server plays the important role of exchanging and broadcasting CAD information from one client to another client. In other words, the server, as a registration system for multi-user application session, is like a messenger between clients. Clients are "equipped with whole CAD systems and some communication facilitators" [22]. In the latter architecture the server side carries out the major modeling activities with visualized-based manipulation client [23].

Our prototype, called NX Connect, is built on a thin server with strong clients. The server acts as a messenger to distribute model changes between collaborating users while maintaining all the necessary information. Each client is equipped with the full CAx application and generates a local copy of the part file that can be updated during the design session. Since the local stations (clients) perform the computations, a real time multi-user experience is delivered.

2.4 Existing MUG Architectures and Infrastructures

Multi-user applications essentially rely on "an underlying distributed communication layer to perform message passing between computers" [8]. Most of the architectural styles found in multi-user systems follow a philosophy where application data (the model) is kept separate from interface code (the view) [6]. Collaboration transparency follows this style. It will not modify the actual application after collaboration support is added to the application [8]. Its systems handle collaboration implicitly and "replicate display output and adopt an approach based on sharing the presentation of an application" [26]. Therefore these systems give each user "a common frame of reference " or a common context, which support the What You See Is What I See (WYSIWIS) idea[26][27]. However, pure WYSIWIS sharing cannot be used on MUG development, especially when multi-users build the same model where different tasks need to access shared data. Different users may require “different presentation–level or WYSIWIS information sharing” or some of the users may consider all levels of presentations. Richard, et. al [26] state that collaboration transparency cannot provide this level of flexibility because the system is required to exploit knowledge of the shared task undertaken by each user. If that is the case, the alternative approach is a collaboration aware application.

In contrast to the transparent approach, the collaboration aware application develops special purpose applications, which recognize levels of cooperation and handle collaboration explicitly. The collaboration aware application provides facilities to “explicitly manage the sharing of information,” allowing sharing to be presented in a range of different ways to different users [26]. This approach often demonstrates how the information is performed and modified within the application, and how the decision of MUG’s management is embedded in the protocol. Furthermore, the collaboration aware applications lack a supporting infrastructure and inhibit interface tailoring of sharing policy, which often require the developer to conduct the application from scratch [26]. So the transparent approach is more popular for developers to apply to MUG applications.

Besides the transparency approach and collaboration aware applications, there are other methods available - blackboard architectures and Groupware toolkits widgets. Gelernter, et al. [3] and Martin, et al. [4] point out that Blackboard Architectures allow “multiple processes to communicate by reading and writing information from a global data store” and provide “a flexible framework for problem solving by a dynamic community distributed process”. Blackboard architectures are “an associative memory with multiple independent cooperating knowledge sources, competing hypotheses, levels of abstraction and feedback to the sources” [11]. Another option is Groupware toolkits widgets. Hill, et al. [3], suggest that Groupware toolkits provide reusable GUI widgets and communication infrastructure components [5]. It also “integrates with a set of popular Integrated Development Environment (IDE)” [5]. The user interface of multi-modal interaction supports “multiple forms of interaction, such as a combination of speech and gesture [6][7].

2.5 Multi-User Awareness and Agent Software

Multi-user awareness is “the up-to-the-moment understanding of another person’s activities in a group environment” [5]. This includes “information about who is using the system, where they are working, and what they are doing” [5]. Gutwin [8] suggests that multi-user awareness is “an important part of collaborative activity. If workspace awareness is difficult to maintain, collaboration becomes more difficult”.

Agent software is a popular and effective approach to increase multi-user awareness between users. Researchers have had experience in multi-user interfaces for many years. Liu, et al. [9][12], propose that independent agent-based methods be deployed to enhance the multi-media communication between collaborating personnel. The intelligent agents can apply rules to communicate among themselves and distribute change information among collaborators. P2P (peer to peer) is “any distributed network architecture composed of participants that make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts)” [24]. Each site has intelligent agent software to support interaction between multiple users. Agent software can transmit the collaborative network and capture audio and video streams locally for each user; meanwhile another agent might record design changes, and update them between users based on users' privileges. Douglas, et al. [15][12], apply an agent-based approach to facilitate real-time interactions between player entities in a P2P game architecture.

Shen[2] proposes Augmented REALITY concepts that demonstrate models between the collaborating team members, but only one member has the right to modify the model. Wallace, et al [9][12], "discuss a multi-user X window application that associates multi-colored cursor software to time-share the system cursors among all the users." User conflicts occur without carefully orchestrated interactions [9]. "Multi-cursor ICEWM has allowed multiple users to work on different applications, but the window manager is still time-slicing single system cursors [10][12]".

2.6 Motivation and Challenges

Many of the multi-user GUI's for the research prototypes that we have identified orchestrate the user interactions, because the interfaces have no flexible means of interpreting the actions among the users, or even regulating how they choose to interact. It is one thing to continually update the screens with all changes among users, but it is another thing to investigate what type of change information might be useful among a set of users and to permit the regulation of that data among the users. It is also necessary to provide an informative collaborative environment for multi-users with awareness information [5]. "Generic, customized, and groupware-specific components" [14] should be contained in the MUG (Multi-User GUI).

Compared to the widgets in a single user interface, distributed communications protocol, and the rendering and visualization of multi-user data are fairly complex in a multi-user environment. First, a multi-user interface needs to express which user is managing a component.

Second, a multi-user interface needs to determine whether a local user or a remote user manipulates the widget [5]. The visualization for remote users is often different from that for local users within a widget [5]. Finally, Hill [5] states that it is a complex issue to “handle simultaneous access to the widgets by multiple users”.

3 CONTEXTUALLY FLEXIBLE MUG ARCHITECTURE

Based on the literature review, previous research demonstrates some product specific MUG's, e.g., the CoMaya MUG is only designed for the product Maya. Generally, these references do not generalize their architecture to account for flexible context preferences. How they would apply transparent methods over different network modes such as CS, P2P, or Cloud Serving is not clear. User awareness issues that consider culture variability and even personality variances are not addressed well in CAx multi-user research.

The thesis will extends previous efforts in MUG's as described in the literature and achieve the thesis objectives as we stated earlier:

1. Investigate and develop architectures for multi-user interfacing of localized and distributed collaborators applying CAx applications, including agent software that can be configured for user context preference.
2. Determine how to integrate filters to provide personalized and selective contextual presentation of collaborating user interactions. A filter provides a function/algorithm to limit collaborating data input, to provide a different view of that data, or to promote multi-user awareness. Filters could be mapped to Multi-User GUI (MUG) menu items or selection icons, etc.
3. Develop MUG prototypes that are consistent in look and feel with current commercial CAx GUI's, but can be invoked or hidden. The intent is to develop software libraries that can be linked to CAx API libraries, and then eventually become a member of the

API library after commercialization by the CAx company. In addition, a prioritization method will be considered that allows manager oversight of multi-user sessions.

4. Demonstrate that MUG's can be practically used to enhance multi-user CAx; and that product personnel of different cultures can navigate CAx collaboration effectively, in spite of cultural differences.

The following sections outline the proposed architecture for a MUG that provides user flexibility to configure the context to meet the interface styles and cultural constraints for each user in a collaborative session. These sections are then followed by sections that describe a limited MUG prototype built on current API's and other software to emulate many of the desired features for the objective MUG. We then compare the desired MUG with our prototype to show current limitations in CAx API's, CAx architectures built for single users, and current computer OS's and associated networking technologies.

3.1 MUG Architectural Concepts

Multi-user collaborative tools are rapidly being developed and will be deployed over the next 5 – 10 years. It will be necessary to conduct research into multi-users engaged in collaborative product development, with interface architectures that acknowledge the cultural variations between users, and with flexibility to specify and manage interaction context as important productivity variables. MUG architectures and applications must be developed concurrently with advances in collaborative networks and collaborative CAx applications. This thesis recommends one such MUG architecture.

Improvements in CAx architectures and their application API's, along with more robust networks, will influence the effectiveness of MUG applications. For example, consider the current Siemens NX API and the lack of an event interrupt. When an important parameter is changed in an NX session, any application built upon the NX will not get immediately notified of such a change, because NX does not throw an interrupt event. This makes automatic change notification between multi-users difficult. This simple limitation, symptomatic of most modern CAx applications designed for single users, limits how MUG users interact. The lack of access to multiple event buffers and multiple screen cursors on a computer also restricts flexibility in communicating change modification between multi-users in the same application model.

Obviously, the speed and quality of collaborative networks will affect the collaborative productivity in MUG applications, but cultural interactions can equally influence collaborative productivity as well. Existing MUG software such as multi-cursor ICEWM [9] and multi-cursor X window proposed by Wallace, et al. [12], do not capture different cultural elements or manage flexible contexts in global development environments.

The proposed MUG architecture is a collection of software and application interfaces that multi-users apply to collaboratively make technical decisions in completing their application tasks (design, analysis, process planning, etc.) over local or global networks. In this architecture users select preferred collaborative contexts to regulate the way they wish to interact with multi-users, or as users are directed by supervisory personnel to regulate multi-user tasks. A MUG interface, as an add-on to multi-user software and CAx applications, is the architectural application.

As an interface layer on top of the CAx application a MUG will achieve the concept of “the model being separated from the view” as discussed in section 2.4. In one implementation the MUG interface is detached from CAx application data. Contextual changes in a user’s MUG will not affect the application data inherent to any Multi-User Software (MUS) used to communicate CAx application data among multi-users. NX Connect is one example of a MUS.

The MUG architecture should also be general enough to integrate into multi-user CAx applications regardless of distribution network type, such as CS, P2P, hybrid forms of CS and P2P, or Cloud Serving. The proposed MUG architecture and associated software libraries will be general enough to be used in most CAx applications.

3.1.1 Context Preferences

Flexible contexts will support information sharing and allow multi-user activities to be reflected on all user screens as desired. Sharing contexts can be set to interpret discourse and communicate modification intent between users. Flexible contexts will provide different sets of circumstances (like video chat environment and text environment) to help users communicate with each other and will be programmed into the future MUG. Users will have a variety of options for communicating with each other and setting the contexts for the communication:

- *Display control contexts* - Each user will set a display mode context, which identifies if other user remote sessions and their screen displays are to be observed locally, and, if so, how and how often. Context preferences will be set by each multi-user to 1) share an entire screen between selected multi-users; 2) share only the CAx workspace screen, not including the GUI; 3) set the timing for refreshing multi-user windows on a user's local display(s); and 4) if the user is the supervisor, set security tokens that limit the model information transmitted between multi-users. For proprietary reasons each multi-user will have rights, from none to full access, to block certain aspects of model viewing by other multi-users.

Effective multi-user display contexts will depend on the viewing environment and related display equipment, being most effective in rooms where large multiple displays can be used to mirror the session windows of networked multi-users. Multi-user display collaboration will be limited on a single display screen, unless the display is very large, or high resolution projectors are used to partition multi-user sessions on a large wall screen.

Data compression and decompressing techniques will be very important to propagating multi-user session windows among multiple users, although techniques such as windows terminal sharing and HP's Remote Graphics Software (RGS) clearly demonstrate that network bandwidths are capable of mirroring distributed terminal displays, also capturing mouse cursor and keyboard events relative to a screen layout.

- *Communication contexts* – The proposed MUG will provide tools to allow multi-users to communicate by 1) video, 2) audio/voice, 3) text, and 4) emotive icon. Each user will set communication context preferences, unless a supervisor overrides the user's context preference. ***Video contexts*** depend on each user having a webcam or other video source, and enabling it for multi-user viewing. Each user will have the right to specify those multi-users to view, and filter out those not to view, unless supervisor override.

Using ***audio contexts*** and related equipment (headphones, external microphones, etc.) a multi-user can enable one or more users to enter an audio session, or allow the user to send voice mail to a multi-user. When combined with video interfacing, this can be an effective way to exchange production information and design details. Modern VOIP and Skype applications abound with this capability already. Researchers at

BYU have embedded Skype capabilities into a Siemens NX CAx application, calling the prototype NX Skype, but the prototype does not include video collaboration.

Audio contexts may not be effective when different cultures are collaborating in multi-user CAx sessions, and located in different time zones. In this case text fields will be provided to multi-users through the MUG. Similar to chat boxes, each user will set a ***text context*** that would include no text communication allowed, or text communications allowed in a language of preference and with a notification context, such as text inbound alert followed by user response, streaming without requiring user response, or a fixed time interval before text viewing or response. In the fixed time interval mode text is buffered for later viewing and response; this mode is important to users located in different time zones. When different cultures are texting, the user can set a translation context which consists of 1) no translation, show in native language received, or 2) translate into current language preference.

Icons will provide a way for users to express themselves emotionally, and provide an effective rhythmic order for users to visually manage applications and improve awareness in a collaborative environment. ***Emotive context*** preferences will enable users to use emoticons in their communication, and to relate them to the audio and texting communications between multi-users. Emoticons within a message can enhance dialogue and save time to lighten the mood. For example, "humor and sarcasm don't come across well in text". [35]

- *Supervisory control contexts* - A future MUG will provide organizational contexts, by providing both a teaching context and a management context. In the teaching context a supervisor/instructor can remote control another user's cursor while tutoring the user. A supervisor can also assign tutor authority to any multi-user team member under restrictions, such as limiting the power distance [34].

A supervisor will be able to limit multi-user model access, including team participation scope, and participating timelines. Some participants will only access part of the team resources while others may have full access. The supervisor will be able to delete a user's account or change the user's role on a team.

- *Security contexts* – Although mentioned earlier, security contexts will allow supervisors to set the collaborative access rights to CAx application model data. Extending participation rights to supply chain personnel who can identify current problems in obtaining material resources, or supply chain limitations in globally distributed design and manufacturing facilities, will counter potential resource problems early on.
- *Recording context* – The proposed MUG architecture recognizes the importance of multi-user CAx for generating product development histories, including critical design and manufacturing decisions. This information is almost impossible to maintain in the current single user application interfaces. Thus, each MUG user will set a record context to record or not record multi-user session data, along with several contextual preferences to designate which of the communication means are to be recorded: audio, text, emoticons, etc. It is assumed that the MUS application will record the CAx model changes and maintain a master product model.

3.1.2 MUG Modules

A MUG is broken into two modules: MUG Client Application (CAPP) and a MUG Information Storage Module (MUG ISM). Each multi-user applies a CAPP to program and regulate the interaction with other multi-users logged into an application session. A CAPP will be deployed differently depending on the network type: CS, P2P, or cloud, but the basic functionality will not change. The ISM functions to store critical user authorization information, e.g., each user's technical background and cultural information at a critical location server. This thesis does not address the security management details for multi-user sessions.

The CAPP provides tools and a GUI to serve two primary functions: 1) program and store contextual preferences for multi-user collaboration; and 2) transmit, receive, and store session data propagated between multi-users. Common terms used to describe these two primary functions are MUG filters and MUG agent software, respectively. A MUG filter is any method, such as a widget application, used to set the flexible context state, whereas agent software can be configured to package and transmit multi-user session data, according to the current filters set by the user. This data would include user contexts, user awareness information (there/not there, active/not active, session type, security restrictions, etc.), filtered application data packets (what application data is transmitted and viewable; e.g., some users may be restricted to viewing only a subset of a design assembly), and network communication speed/stability data.

The MUG CAPP constructs network methods to process MUG applications. MUG CAPP includes a variety of widgets, which are small applications that can be executed within the MUG and allow users to configure an existing applications for the user's preferences. For example, A Radar View Widget can reflect a single user's activities on other user screens. A Text Widget can provide text display and allow users to display and edit texts. A Translator Widget can bring real-time, in-place translating services. A security key widget can ensure all files are transferred in a secure environment.

Each user will use their GUI CAPP to connect to other users. The user information data is stored by the MUG Information Storage Module and can be quickly accessed by other users' MUG. The number of new features for managing interaction contexts will increase with advances in collaborative networks and collaborative CAx applications. The development of new features also depends on continuous customer needs. Complex visual representations of multi-user awareness and distributed groupware infrastructure will advance in the future.

The MUG Information Storage Module will store authorization information and user identification information. This module interfaces to the MUG CAPP of each client and stores or retrieves data as needed by the client's MUG CAPP. The module stores MUG context–related information and organizes data types and user information.

The MUG architecture is general enough to be distributed in several network architectures: 1) CS; 2) P2P; or 3) cloud server. These will be considered in the following sections.

3.1.3 MUG in CS Mode

Figure 3-1 shows how the proposed MUG components are deployed over the Server and Client in the CS mode when multi-users engage in a collaborative Multi-User Software (MUS) application. Presently, MUS prototype software such as NX Connect acts as an application layer on top of single user CAx applications, where API calls are used to observe, record, and transmit data modifications among multi-users. Future versions of CAx applications will integrate this functionality directly into a specific CAx application interface.

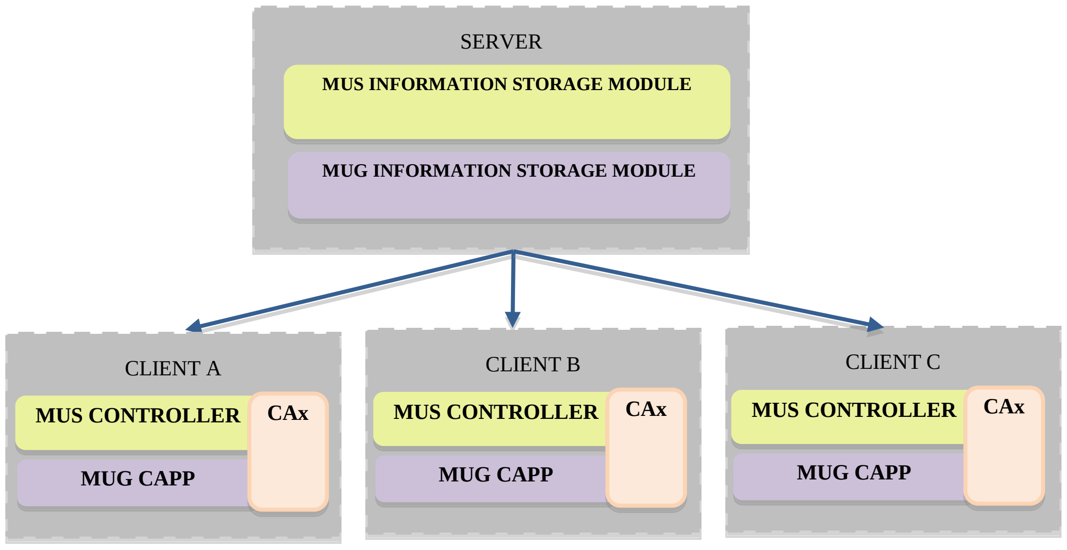


Figure 3-1: MUG Architecture CS Mode

A server will include both the MUG ISM and the MUS ISM. But at each client computer, a MUS Controller will convert design modification information into primitive values and transmit them to the MUS ISM. The data will be stored and transmitted to multi-users according to permission and context settings. At each user station the CAx application will construct corresponding features in their CAx application, depending on the permission settings for each user and the desired contexts.

There are some important questions to be answered in the future. Where should the context filters and permission filters for each multi-user be applied - at the user computer, at the server, or at both? Should all multi-user session data be sent to the server, where each user's context and security filters can be stored and thus applied? This method permits all multi-user and application data to be recorded at a single server, enhancing session security, and minimizing data hacking at a user's computer, but will certainly increase data network density and server data storage requirements. Conversely, applying filters at each user's computer to strip down the data packets will mandate more sophisticated and localized security software to protect proprietary data from unauthorized viewing/storing, but minimize the data to be sent.

The MUS controller function is critical. If data capture and data synchronization in the controller are mainly processed at the client side, as shown in Figure 3-1, a CS with a thin server and strong client will be utilized. If the data related to multi-user activities are computed on the server side, a strong server with thin clients will be utilized.

The MUG will be able to layer itself onto or be integrated into the MUS and CAx applications. The data related to the MUG application will be recalled and stored by the MUG Information Storage Module.

3.1.4 MUG in P2P Mode

Figure 3-2 shows how the MUG components are deployed in the P2P mode. Peers will function as both clients and servers to other peers and make the information of the MUS and MUG applications directly available to them as shown in Figure 3-2. MUS and MUG ISM's will reside in each peer computer and be linked to each other in the overlay network. Each peer will employ an interfacing protocol or flood through the network to route a search to other peers for multi-user session data. In P2P networks, once a network opens up to the public domain, users begin to access a portion of resources. The more peers that join the network, the more resources become available to each peer. This may cause team-shared files to be unsecured. For security reasons it is recommended that important user information and part files be stored in a more controlled depository.

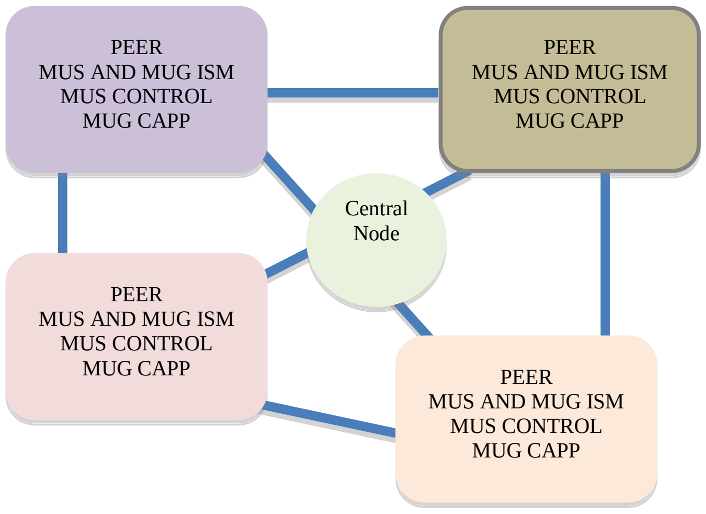


Figure 3-2: MUG Architecture in P2P Mode

For example, an ideal P2P architecture may not be secure for the data storage requirements of a complex CAx collaboration. It seems there has to be a central infrastructure node, which serves to enforce security and store the "master changes", even though the MUS and MUG are deployed over all the peers. This central node might be a supervisor's peer computer or any other peer computer so designated.

In an ideal P2P case all data is passed among the users and filtered at each peer computer, including sensitive data that should not be viewed by all multi-users. The central node infrastructure, which deviates from the ideal P2P, would store sensitive data in the central node and not allow it to be generally distributed to the peer computers. Each peer could then remotely access this data but not make a copy in the local machine. The central node will pass and filter the regular data as other peers but have a higher data storage requirement to process and store sensitive data.

3.1.5 MUG in Cloud Serving Mode

Cloud Serving is broken down into three services: “Application as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS)” [36]. Customers will choose an option within their comfort zone based on a different purpose and/or budget**.** The MUG ISM and MUS ISM will be stored on several virtual servers hosted on a large data center, which is labeled as “Database” in Figure 3-3. MUG CAPP and MUS applications will be stored in cloud application servers that are labeled as “Application Servers” in the figure above.

Typically “Application Servers” have many blades, each of which can host applications. “Central Nodes” in the figure will be programmed to monitor the network traffic and administer the system to make sure everything runs properly by reporting event information through the collector component to the server. Protocol and Middleware used in “Central Nodes” will set rules and maintain communications among the computers in a network. “Server Blades” will handle all the heavy workload generated by the running applications. Client computers functioning as display terminals will access MUS and MUG applications remotely through interface software like Remote Graphics Software (RGS). Where a MUS application is replicated for each client, a MUG would also be replicated for each client. Multi-user clients see the same application interfaces in Cloud Serving as they see in CS architectures.

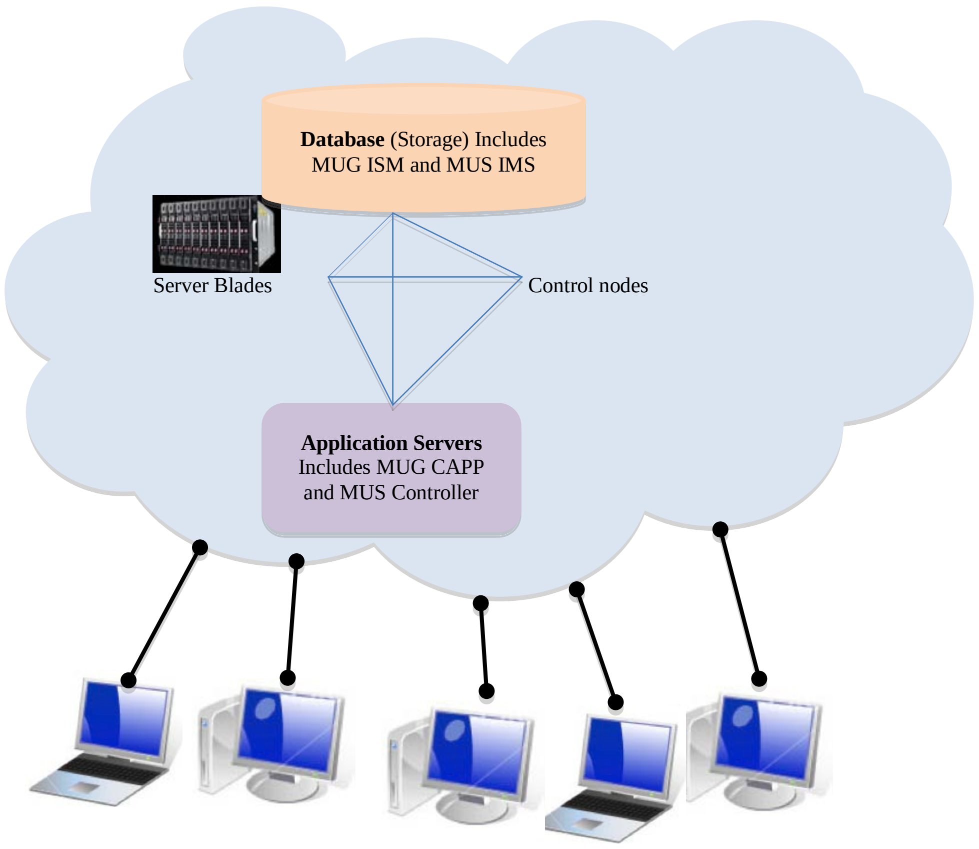


Figure 3-3: MUG Architecture in Cloud Serving Mode

An interface like RGS is used to connect client computers and blade servers shown in Figure 3-4. RGS sends user mouse and keyboard events from client computers to blade workstations. These events will be interpreted by RGS software and processed by the applications on the blade servers. RGS also transmits the interactive desktop images running on blade workstations to client computers via a TCP/IP network.

The Session Allocation Manager (SAM) optimizes the deployment of client/application sessions among the blade servers. Presently SAM only allows the user to log into SAM once or establish a re-connection to the user's previous remote computing resources. SAM also manages connections between client computers and remote computing resources (like Siemens NX, MUG CAPP, MUS controller, etc.). The database in SAM will store the properties of the computing resources, client computers and RGS and will determine how to allocate user requests based on the computing resources.

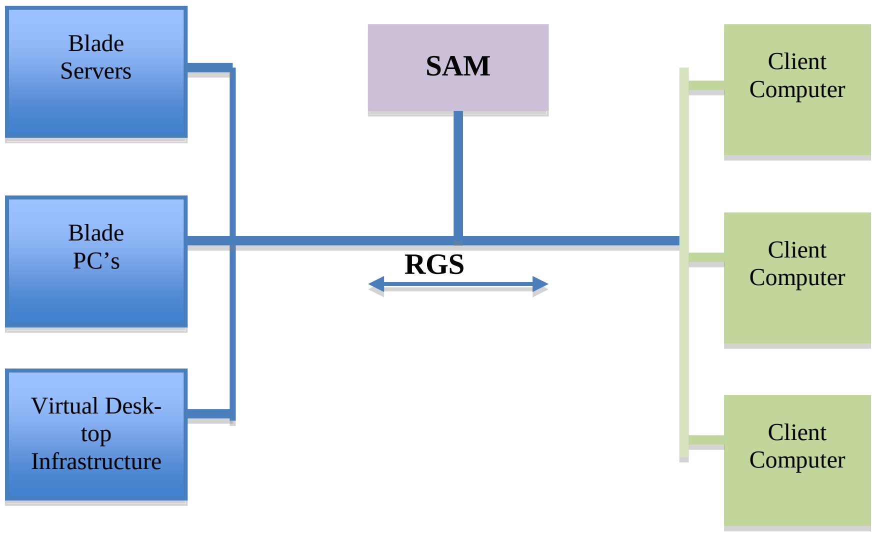


Figure 3-4: RGS Architecture (Source: HP Website)

Security will be enhanced due to the centralization of data, and the performance will be monitored and scalable. But users will lose control of local data and the performance of CAx applications will be greatly influenced by high network load or low bandwidth.

3.1.6 Agent Software

Agent software is programmed to specify how one user interacts with other users and exhibits a high degree of autonomy. It is configured to observe, record, and transmit user actions among a set of collaborators. Intelligent agents apply rules to communicate among themselves and distribute change information among collaborators [12]. In the proposed MUG architecture, the agent software uses the context settings to determine what information is to be transmitted. Again, an issue to be resolved in the future, and which depends somewhat on resources, is whether the agent software strips the data packets locally, depending on context settings, or whether the agent software transmits the context settings to a server to strip the packets before distribution to clients.

The advantages of agent software are: 1) simplifying the complexities of distributed computing and 2) overcoming the limitations of current user interface approaches, such as speed of performance, error rates, and retention over time [31]. Agent software will periodically check if any information has been changed and then broadcast updates to all users.

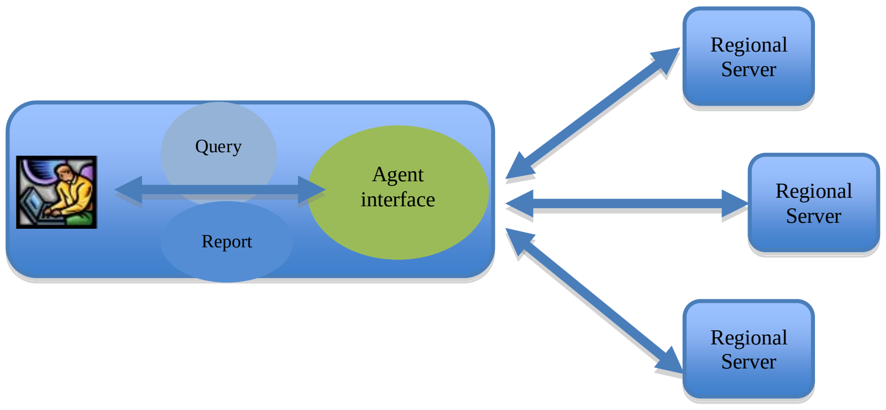


Figure 3-5: Agent Software Workflow

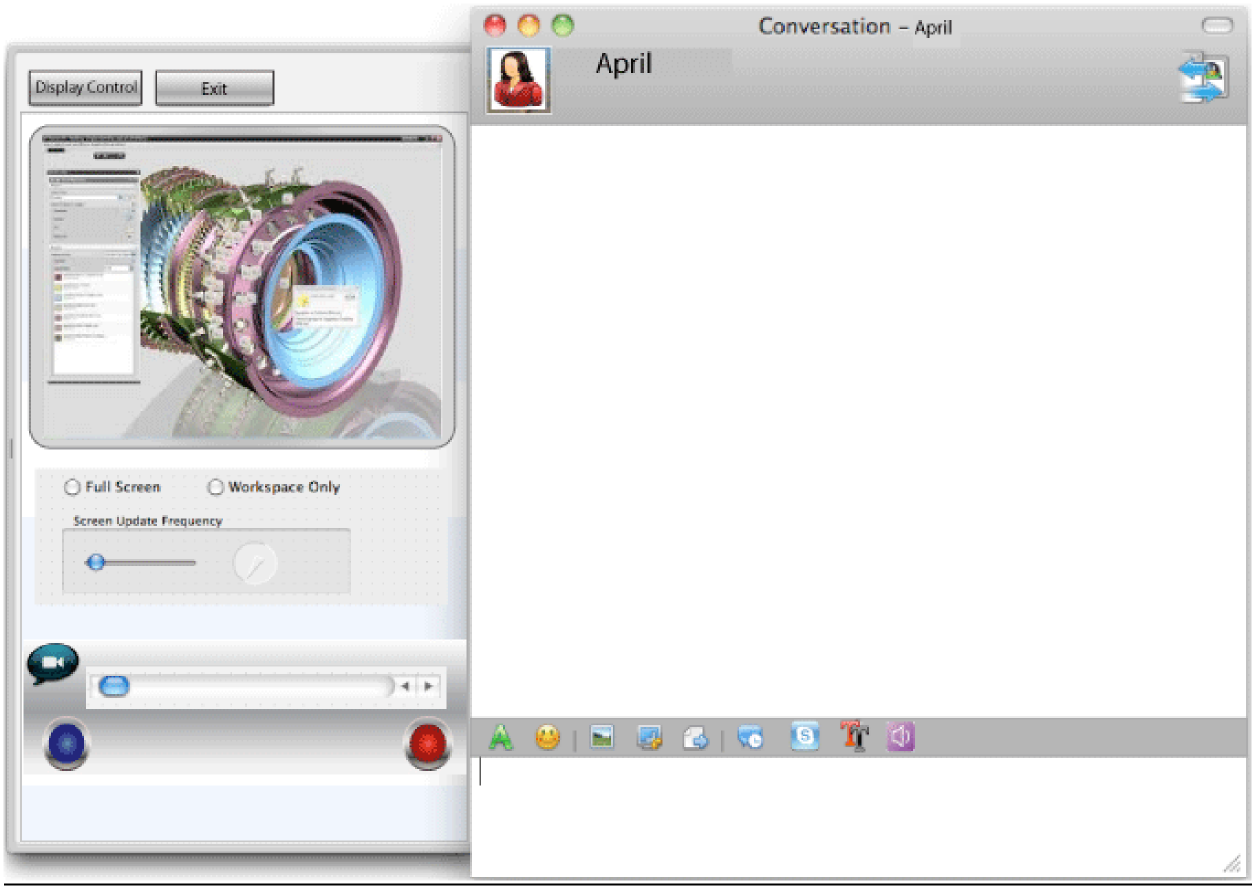
The agent interface can receive a query regarding the collaboration environment, such as context settings for an on-line user, and report the query results to the users shown in Figure 3-5. In the CS mode a regional server, at some remote location, receives requests from the interface agent software and collaboratively responds to the request.

One useful feature of agent software is to integrate video and audio streaming systems that can be broadcast to the set of collaborating users in a MUG. The agent software may use any of the architectures previously discussed, such as the P2P and CS architectures. CS can handle account management and present status information. P2P is used for in-band real-time streaming data that includes audio, video, and text messaging. It is scalable and robust because the removal of nodes has no great impact on the network.

3.2 Contextual Filters

As shown in Figure 3-7, the future MUG will contain more features associated with a serial of filters. The translator will be integrated into NX MUG and demonstrate the translation outputs in the same window. A user can pre-set the mother language and target language, type his mother language, and press the translator icon, changing the content into the target language in the same place. There will be a frequency slider, which allows a user to choose how often he wants to see another users' screen updates. Also in the future, a region selection radio button will be added in the MUG window, which allows a user to select the different regions of workspace he wants to show other users.

A user will set on-line status as away, busy, or on the phone, or temporally block users. If a user signs in as a supervisor, he will be permitted to control of user cursors for instruction and tutoring.



Text	Emoticon	Send Images	Capture Screen	Send Files	Message Files	Skype	Translator	Alert	Recorder Mode	Stop	Start

Figure 3-6: Future MUG

3.3 MUG Prototype Considerations

In this section we configure our prototype, considering the limitations in current CAx API (and in the Windows OS) and other tools that are available.

3.3.1 CAD API

A CAx application provides API's to extend the design environment, enabling engineers to design and produce better products. Zhou et al. [28][30] used "the APIs of multiple CAD systems in order to develop a CAD neutral module which will be able to interact with the different commercial CAD packages enabling more collaborative design efforts to take place." Kao and Lin [29][30] used "the CAD API to pass model information over a LAN as well as the Internet as part of a collaborative CAD/CAM system in development."

A CAD API also helps the system extend capability and compatibility. Tucker's research involved the development of an API translator, which was able to convert API functions and code from one CAD system into functions and code for another system. Tucker's research demonstrates a partial solution to "a pervasive problem in CAD which is the collaboration and file sharing between groups or companies which use different CAD packages" [30]. He developed an API translator, which was able to convert API code and function for one CAD system into code and functions for another CAD system. It is important to design an adaptable and user-friendly platform. We suggest that a future MUG would develop a collaborative library that could be integrated into the API of any commercial API library.

3.3.2 NX Connect

NX Connect is an add-on MUS to Siemens NX and allows multiple users to access and make changes to a single part file simultaneously. It is a collaborative CS prototype without a multi-user MUG; thus, all collaborative activities in NX Connect have to be orchestrated. Before building an NX MUG as a demonstration prototype, we need to understand the NX Connect architecture.

3.3.3 An Overview of NX Connect Architecture

In the NX Connect multi-user CS architecture the server captures and records the design process, whereas the "client generates a local copy of the part file and utilizes change information to locally update the part" on the local workstation [12]. The NX Connect architecture, represented in Figure 3-8, consists of a Data Capture Module, Data Sync Module, Information Storage Module, and NX controller.

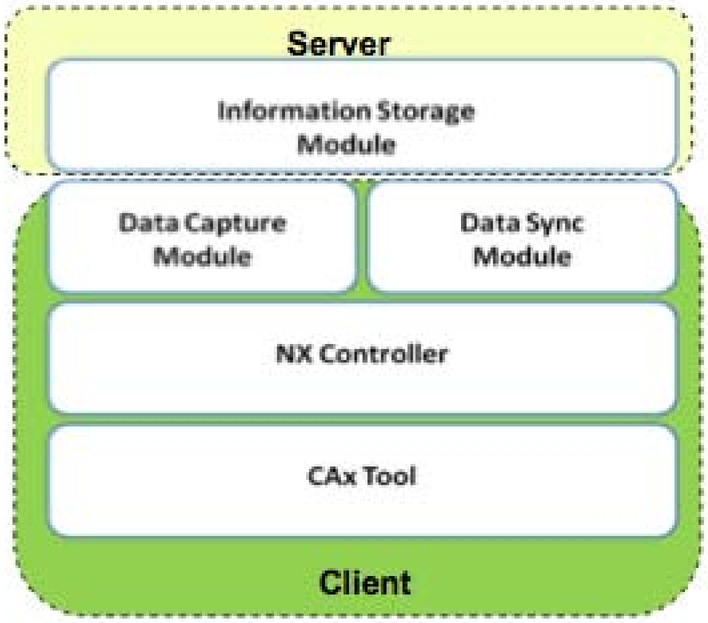


Figure 3-7: NX Connect Modules

The function of the Data Capture Module (DCM) is to capture the design process data including mouse actions, screen shots, undo files, and feature information that has been created, modified or deleted. The CAD API is used to retrieve the status of a part file at any given time [13]. It will limit the amount of captured information and store only that information necessary to recognize the changes that have been made to the part file and whether an actual action was implemented by a user.

The function of the Data Sync Module (DSM) is to update the NX part file from part model changes on a different user's computer and to update the model periodically. The relevant actions are determined by the Information Storage Module, which draws inferences from that data to limit the information captured by DCM. The database is updated and queried by a specific timer "set to a specific wait time of 1/10th of a second" [13], which is the polling rate. Polling is used because the Siemens NX API does not user event interrupts when a model parameter is changed.

The Information Storage Module (ISM) stores all model change information including the action objects, part features, and related information. The primary qualification for designing an ISM is that it must quickly and efficiently store and recall change data [13]. Microsoft SQL and the enterprise class RDMS is used to implement the ISM. A series of tables are created to organize different data types and manage the session and user information. These tables are linked together and are accessible through the SQL languages. Each part is assigned a Part ID, which is used in the child table that contains all the features. In the features table each feature name is associated with a feature ID that is utilized in the next child table corresponding to the feature types (sketch, extrude, united, etc.) [12].

The NX controller is the core of the NX Connect architecture and consists of two main functions – push and pop. The NX Controller receives data from the DCM and DSM, then "converts all the part information into their primitive values and stores them in the database, as well as converting these parameters back into useful data to construct the corresponding features in the NX session on each user's machine" [12].

In the push function, when a user modifies a parameter in the model and the DCM reports a change in the local NX part file, the controller collects all relevant information about the modified feature. The NX controller utilizes API commands and determines the feature type and then deconstructs the feature into the basic parameters, like extrude. The controller will store the specific feature name for later reference. All such information is sent to the corresponding table in the database and it is distributed to each user.

In the pop function, when a user modifies the existing CAD document, and changes are uploaded to the database, the DSM will pop up an alert to the Controller. “The NX Controller will then read the feature data and place each parameter in the appropriate API function to recreate locally the feature or modification on all user workstations” [12].

3.3.4 Limitations of Siemens NX

NX Connect is limited by the single user architecture of Siemens NX, as follows:

1. There is no direct access to event handler and undo marker files are utilized to signify major changes in a multi-user environment.
2. API’s provide handles to geometric objects (memory addresses). These cannot be passed to other multi-users since memory locations on each computer vary. Thus, NX Connect required extensive programming to extract the parameters from the data structures. CAx applications inherently know how to pass geometry parameters because they generate files that conform to IGES/STEP standards for neutral file interchange. Multi-user CAx will need API’s that provide the parameters directly (like copying the object), rather than just the memory addresses to the object).
3. CAx applications have a concept of a single display screen, a single viewpoint, and a single cursor. The NX GUI’s are built for single users and are not intended to be shared by multi-users. Only one user can modify the model at one time, regardless of how many are viewing the display screen. Users cannot simultaneously view the

model from different viewpoints or simultaneously edit the model parameters in the same GUI.

4. There are no event interrupts available to parameter changes inside NX; this means we have to poll for parameter changes from undo marker file. NX Connect uses a real time polling service with enhanced synchronous technology because Siemens NX does not use an interrupt through their API to signal model parameter changes. NX Connect must repeatedly check for updates and poll the application for changes rather than use interrupts to notify of model changes. Polling is CPU intensive, which wastes computational cycles in checking the polled data. Although polling is very simple, it would be more efficient to replace polling with interrupts. The interrupt will be raised only when the new update occurs and will avoid checking for new data at regular intervals. Furthermore, busy-wait polling is perfectly appropriate in a simple single-purpose system, but in a multi-tasking system, especially in real-time computing, interrupts are preferred [32]. Interrupts can keep the computer from busy-wait polling, switch a context to an interrupt handler, and change in execution. Interrupts also make it easy to add new functions such as a sound alert when the server receives a new update and sends it to each client.

Asynchronous technology enables real-time collaboration in a same time-different place mode, but it can create communication challenges due to different time zones, and requires great bandwidth and high cost. The problem of asynchronous technology can be mediated by using interrupts. “Interrupts allow the CPU to deal with asynchronous events” [33]. Asynchronous technology is helpful to sustain the history of a group’s activity, and capture collaborative information through the different time-different place mode.

3.3.5 Limitations of NX Connect

As discussed in 3.3.2 and 3.3.3, NX Connect mainly replicates the model at each user’s computer and uses changes to update the model at each multi-user. However, there are two limitations of NX Connect, which makes it inefficient to process data and update changes.

1. NX Connect uses *System.Windows.Forms.Timer* to trigger events in a single-threaded environment. However, using *System.Threading.Timer* instead of *System.Windows.Forms.Timer* is more efficient.
2. NX Connect contains a tool called VisiHistory, which “presents the user with a video of all of the key events that occurred during a previous users’ design session”[13]. The video is recorded by continuously capturing images at a short time interval and compressing and decompressing the data. NX Connect is programmed to render relatively large resolutions of pictures for a small display size, which is not efficient.

As noted earlier NX Connect uses an adjustable timer to determine when the information is captured and is triggered every 1/10th second to check if there is data available to be read. The timer will periodically poll for an event and update changes, and the UI thread will stop pumping Windows messages, being bogged down by handling the invoked requests. This will cause delayed event handling, especially for the WinForms Application (windows based applications in the Windows.Forms namespace), and NX becomes less responsive to the clients when the process of NX Open becomes busy. If we want to use the timer function, the control of the timer in the WinForms is not as efficient as the timer in a multi-thread API.

To understand the reason for Winforms inefficiencies, the differences between the background and foreground threads are introduced below. In a multi-threaded application, managed threads are either background threads or foreground threads. The applications automatically end and exit without considering whether the background threads (non-UI threads or thread.IsBackground = true) has finished or not, which is different from foreground threads. In other words, the application will not stop until foreground threads are terminated first. Once an application is terminated, the background threads stop as well. If WinForms uses multi-threads, background threads cannot directly visit any UI elements in the UI thread; instead, background threads must delegate a method, deliver and transfer the messages to UI threads after visiting UI elements. Timer is the control in Winforms, which executes based on the mechanism of the timer in the Windows Operation System and is not a real multi-thread. On the contrary, there is a timer class in multi-thread API, which runs based on the mechanism of multi-thread. They both act the same, but follow different routine procedures. As mentioned earlier, the timer in multi-thread API is more efficient and cannot block UI multi-threads.

Nevertheless, the timer measures the performance of an application of a low-level technology development and the control is using NX Open for network communications. The timer should be built into a class that contains the sockets and other components, and also manages connecting and disconnecting from the server. So every instance of the class has its own timer so that it can make more than one connection at a time. The class also has a delegate and event handler that will be raised when the data is available to be processed.

Another drawback is that the method of screen capture is not very effective. Assume there is a 1-foot by 1-foot chess board with 64 squares. Each square represents one pixel and its resolution is 8 pixels by 8 pixels. Advance the hypothesis that the monitor has the same color display and there are 2 different kinds of resolution images that are 16 by 16 pixels and 8 by 8 pixels. A camera with more mega pixel capability over the same area of the chessboard requires more space in terms of memory. The higher resolution image takes up a large amount of memory space, as the memory card has to assign each pixel with a color. So the more megapixels in a picture, the more memory it will take up on the memory card. Since the pixels for each picture are fairly large, these pictures take up quite a bit of memory and slow down the GUI rendering rate. If we reduce the pixel size of JPEG files, it will be comparably better; however, it will not solve the fundamental problem. The recommended method to achieve great efficiency is to integrate a third party video capture tool into the current NX Connect. These video capture tools can compress the data files and filter the multimedia tasks in the processing of the data. There are several screen capture tools available for free in the market, like DirectShow, which include a variety of open source filters like DirectX SDK to support video/audio type and allow developers to transform video images.

Incidentally, RGS methods may be a better approach to display the CAx session screens of other users, but there would need to be substantial extension of RGS methods. RGS also presumes a single user attached to a server. RGS knows how to efficiently pass display information over the Internet without need to screen copy into JPEGs. The next section will cover image rendering in more details.

3.3.6 Image Rendering

A future MUG must update the model or assembly component reliably, as multi-users make model changes; otherwise, intent and awareness confusion will lead to design errors. In CS architectures where the CAx application resides locally on each PC (or workstation), screen updates will only be as effective as the network speeds, and may be affected by network latencies and reliability issues. LAN networks within companies will generally be fast. Image rendering within company network infrastructures will likely be very effective and allow multiple personnel from disparate organizations to collaborate.

Multi-user groups that are globally distributed may have to user buffering methods to update changes periodically, rather than attempt real-time screen updates. The proposed MUG allows users to set timing contexts and thus CS architectures should support multi-user collaboration effectively in the near future.

A cloud serving architecture will also depend on network latencies, and may be ineffective across widely distributed global networks, unless the cloud servers are also distributed and updated accordingly using update timers and buffers. Cloud serving collaboration will be very effective local infrastructure networks, and provide for enhanced security.

Software like RGS minimizes the information to be transmitted from the user station to the cloud server, since only the screen pixels, mouse cursor, and keyboard events need to be transmitted. Compressing/decompressing algorithms at the client and in the cloud minimize the data packets to get realistic performance.

If NX Connect and the prototype MUG are distributed in a cloud, using RGS display updating methods, the current thin-server-thick-client CS architecture will be dramatically changed. This architecture will make the data more secure and a centralized architecture will reduce management costs by consolidating resources to one-point management.

4 PROTOTYPE METHODS

The objective of our MUG prototype is to assist users to communicate with each other through NX Connect when multiple users are making changes in a one-part file. Although the methods discussed in this chapter are implemented in NX6, the method should be general enough to apply to any similarly capable environment. The prototype implementation described in Chapter 5 was developed using Visual Studio and NX Open API in C#; thus, this thesis may use language specific to that programming. However, the methods that construct the prototype will be used to extend the basic MUG concepts into any software application where users may wish to collaborate.

4.1 Methods of Building NX MUG

Based on understanding the NX API, NX Connect Architecture and the illustration of Figure 3-8, there are two methods of building a prototype MUG. The first method is to build a MUG inside NX Connect. The current NX Connect has a minimal GUI for loading a part file to a multi-user workspace, and includes SmartHelp and VisiHistory (NX Connect tabs). A MUG could be integrated into the NX Connect application and share the same SQL, Database and li-brary.

The second method is to create a separate program that contains all the features of the prototype MUG. The reason for doing so is that the prototype MUG is mainly made for commu-nications purposes, such as to capture the workspace of each user, to provide communications tools and translation function. The new program will use software agents as mentioned in 3.1., which will support interaction between multi-users to ensure that the individual contributions are relevant to the group activity as a whole. Agent filters inside the software agent can be used to navigate CAx collaboration effectively in the perspective of different users with different cultural backgrounds.

As mentioned in Section 3.3.5, the architecture of NX Connect is not very effective for rendering JPEG files. The prototype MUG transmits multi-user displays periodically as JPEG images; thus, the second method will be preferable.

4.2 Using Socket Application

The popular method of network communication during the design session is to use a socket–based client server application. NX MUG as a communication-based application will often be sending and receiving data, and a network socket has a mechanism for transmitting data to the appropriate application thread across the network. Therefore, the network is illustrated below.

There are two implementations: socket-server and socket-client. The socket- server is implemented in a class, which consists of a main Socket object and an array of worker Socket objects as members. The main socket object receives the messages for the clients. Its task is to forward the transactions related to the specific client to a worker socket when a client is connected. Then the main socket returns and continues to receive messages for other clients. In other words, the server uses particular objects of a worker socket to send data to each client. The socket client application contains a socket object. When a client sends the data to the server, the client will write the data to the socket and send it to the server.

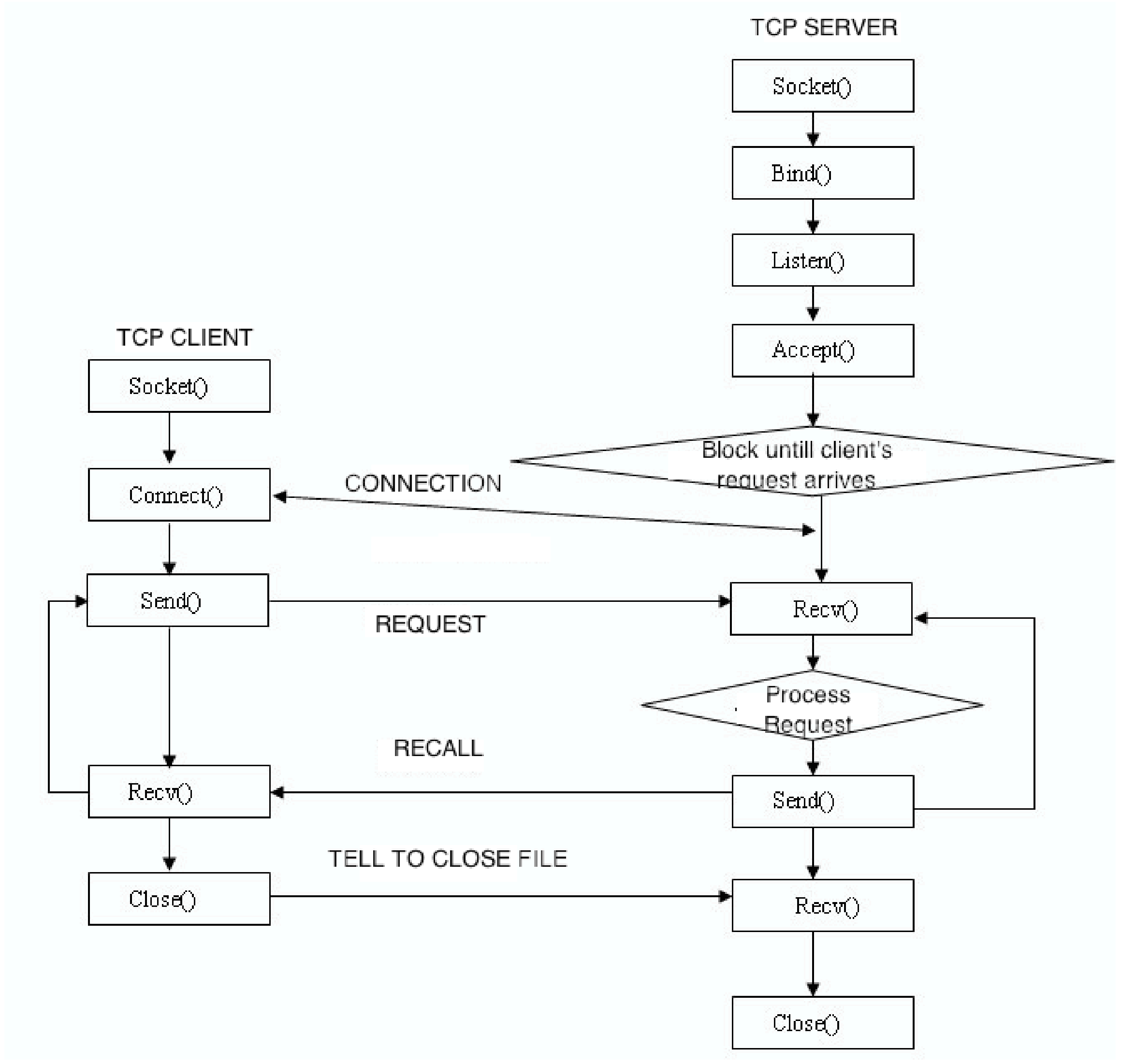


Figure 4-1: TCP/IP Socket Workflow

Once a socket is created, the socket object will be able to connect to the server that is specified by a client, send the data to a remote server, and then receive the data from the server side as shown in Figure 4-1. In the server side, the programmer uses the Bind method to bind the interface in order to link the socket to a local end point and listen to the request from the inter-face. When reaching the client's connection, the socket will call the Accept operation to complete the connection and then a new socket is created to handle the incoming connection requests. After using the socket, the programmer is required to disable the socket and close it.

The option of selecting a socket is determined by the communications mechanism of the server. In general, there are various communication methods between a server and clients and it can be divided into two categories according to the protocol type: one is based on the HTTP protocol and the other is based on the TCP protocol. These two protocols are based on the Web Services Development of SOAP. In fact, the HTTP protocol is an application-layer protocol and the HTTP protocol is based upon the TCP protocol. In other words, High-level protocols, such as HTTP and P2P, depend on TCP ports to allow outside nodes to communicate with specific services.

Besides that, UDP socket is widely used in communication tools. For example, QQ, which is an instant messenger, uses UDP protocol and the advantage of UDP protocol is primarily speed, which does not require overhead to detect reliability and requires less process in the transmitting and receiving of hosts. If you are a regular user of MSN and QQ, you easily find that the better option of sending files is QQ. But UDP is an unreliable protocol and has non-effective fault tolerance. Therefore, it requires writing a lot of underlying code to ensure fault tolerance, which will increase the workload.

Because the MUG will contain heavy communications capacities, the option of building the socket becomes critical. Based on the introduction above, the template method is UDPsocket on the client side to transmit and control the data.

4.3 Architecture of NX MUG

NX MUG uses agent software to connect to NX Connect and the multi-user clients. To link NX Connect to NX MUG the programming codes of NX MUG were placed so that NX MUG is invoked right before entering multi-user mode. In other words, NX MUG will be invoked right before entering multi-user mode. This action assures that NX MUG and NX Connect use the same server and serve the same group of clients. User awareness data and filters of the NX MUG, which organize the MUG by processing the data according to specific criteria, will be parallel to the current NX Connect.

The system will achieve several major functions as shown in Figure 4-2. The function of capturing remote desktop video is accomplished by continually calling Windows system API at a fixed frequency, capturing screen images (or workspace images), and grabbing the mouse movements. The Mouse is drawn up on the images and is saved in bmp format; the images are compressed through the compression algorithm and are converted into jpg format. Then the captured images are transmitted over the network in the segmented form of a byte (input/output) stream. The other end will receive the data packages of a byte stream and assemble them to a JPEG file. Finally, these images are displayed in the video window box.

Another function is sending the chat messages and it is the same as sending pictures that are all transmitted via UDP as shown in Figure4-3. In details, this information is first stored as a data I/O packet that includes other authentication information. After transmitting through the UDP Socket, the receiver will extract the data and the users will see the display sequence in the remote view window and receive text messages.

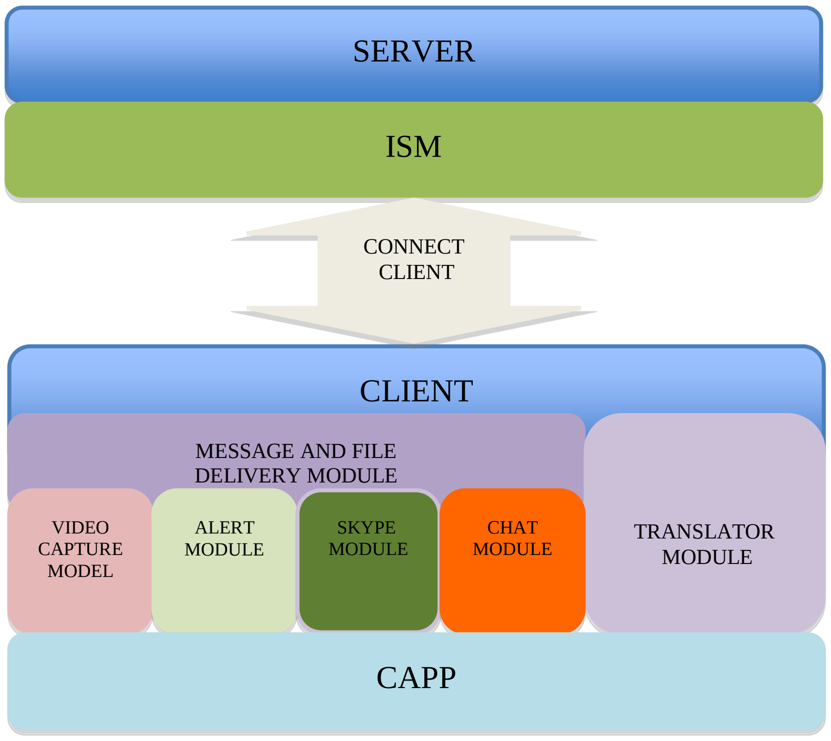


Figure 4-2: NX MUG Prototype Architecture

The alert function has two modes, a broadcasting mode and an alert mode. In the broadcasting mode a message is sent to the server and the server forwards the message to inform all clients to submit the update information. In the alert mode an alert is sent to the target person in the group, which is similar to the principle of sending chat messages in the form of packets as shown in Figure 4-3.

The translation tool is designed by polling Google Translator and it is run by C# web-based tools. The SKYPE function is linked to the program that is written by another graduate student [13]. He integrated Skype to NX.

The functions listed above are controlled by the MUG CAPP.The server only plays an interactive role, which is responsible for reading the corresponding information from the database table of CurrentUser and CurrentUserProfile. The server interacts with the clients to control user login and registration, including distributing of user status, establishing UDP connection for different clients, and broadcasting user's updated information.

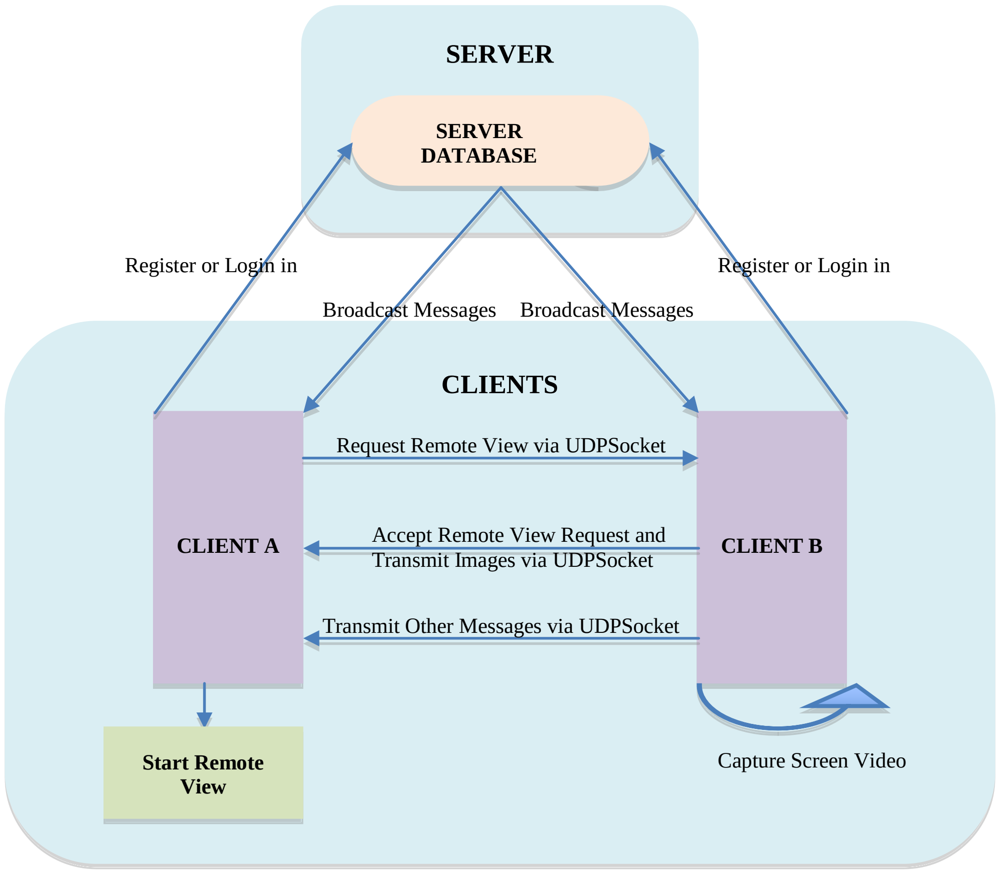


Figure 4-3: NX MUG Workflow

5 IMPLEMENTATION

This chapter a prototype that implements the methods of Chapter 4. The NX MUG application is started by clicking its .dll file. A registry window is launched for the first-time user as shown in Figure5-1. All the information must be filled out; otherwise, a pop-up alert will remind the user to complete the registration form first. In the registry window the Skype information is required to be consistent with same username and password used in the Skype system. To use Skype in NX MUG a user needs to pre-install Skype in the computer. The Project Password on the left column of the figure is usually assigned by an administrator.

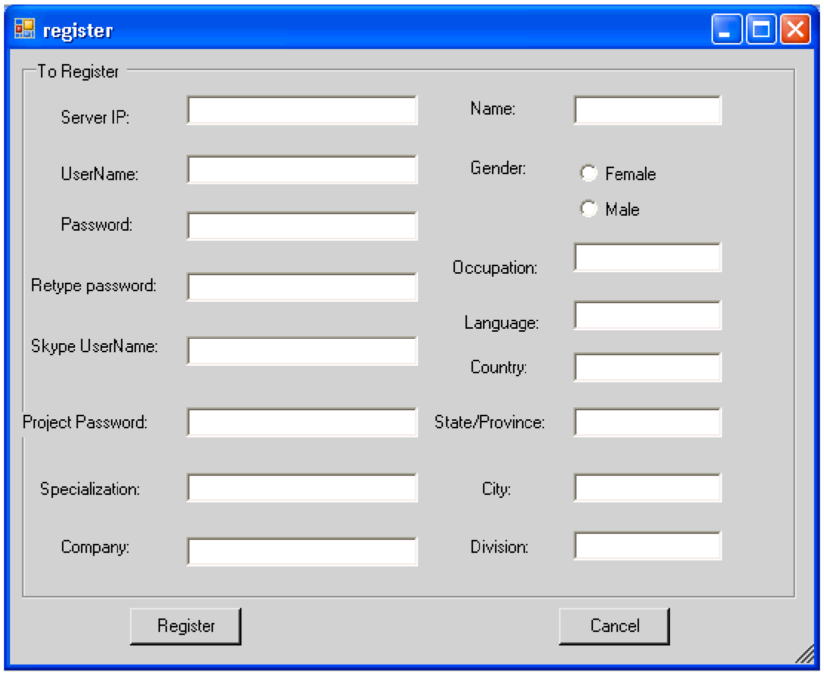


Figure 5-1: NX MUG Registry Window

Then a log-in window will pop up as shown in Figure 5-2. Notice that the next time a user launches NX MUG, the registry window will not be re-launched. Figure5-3 demonstrates the registry-login flow chart. After logging in, a NX MUG main window is launched as shown in Figure 5-4, which lists all the online users.

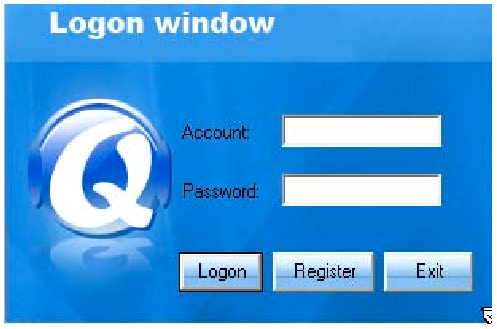


Figure 5-2: Login Window

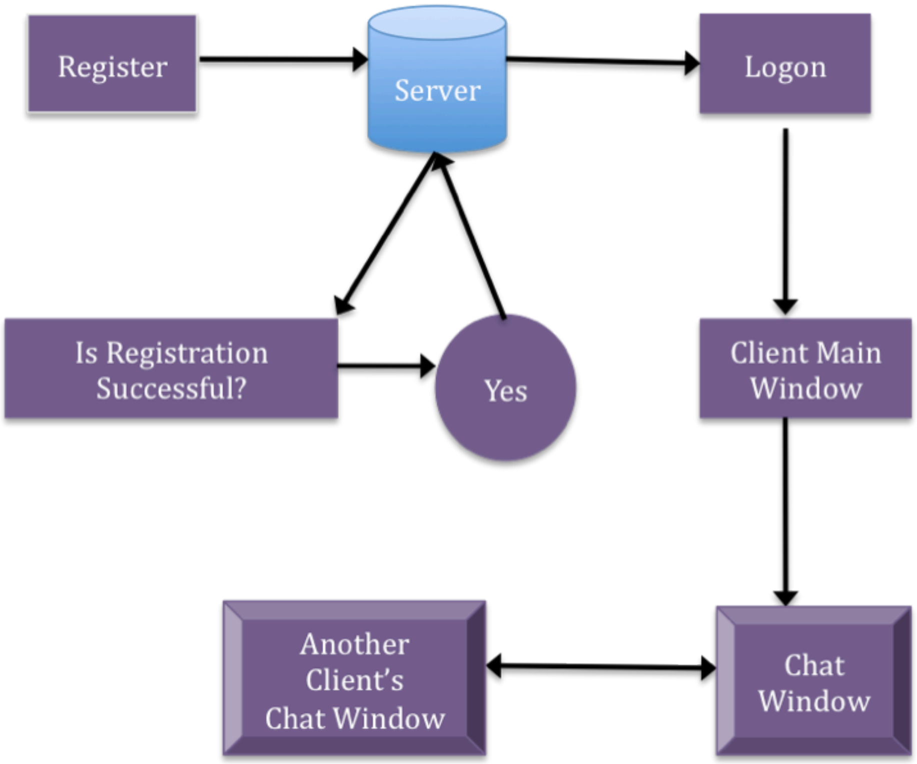


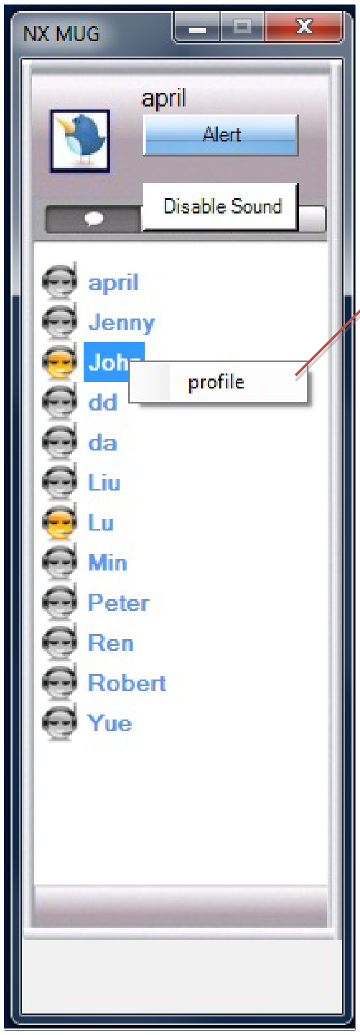
Figure 5-3: Registry Flow Chart

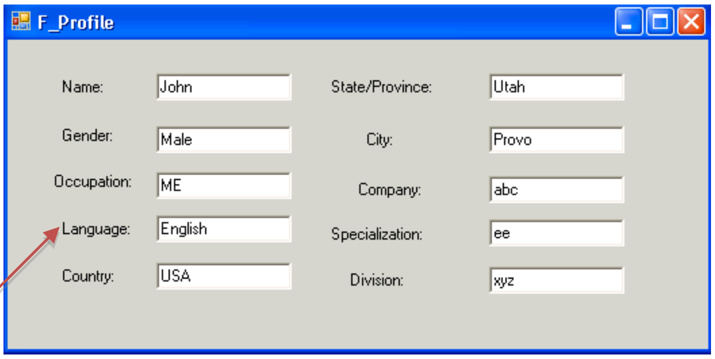
The main window of the multi-user GUI is similar to the QQ layout, which is a form of Instant Messaging. This window can be invoked or hidden at any time, making the workspace available as needed.

The main window contains each client's profile as shown in Figure 5-4. Each client uses an icon associated with a profile to represent each role, and the profile describes the role of each client in the project and which language each client is proficient in as shown in Figure 5-5.

If you wish to follow the work of other other users, their workspace can be reached by inviting them to join in the remote view. When any clients' icons are clicked, a window will pop up. The pop-up window includes a text box as shown in Figure 5-6 on the right. To minimize design conflicts, users can always chat or text to each other. There are three buttons: Alert, Skype and Translator. The translation function will assist team members from different cultures to text each other. The alert button sends an alert to a user(s) after making important changes.

There is a remote view window on right side of the pop-up window. A user can choose to view another user's full screen or workspace by toggling the radio button. For example, when User A clicks on the "Remote View" button and wants to see User B's workspace, a dialogue box will pop up on User B's screen and ask for permission. If User B hits the "allow" button, User A will immediately view User B's window. The pop-up window can be invoked or hidden at any time.




F_Profile

Name:	John	State/Province:	Utah
Gender:	Male	City:	Provo
Occupation:	ME	Company:	abc
Language:	English	Specialization:	ee
Country:	USA	Division:	xyz

Figure 5-5:NX MUG Profile

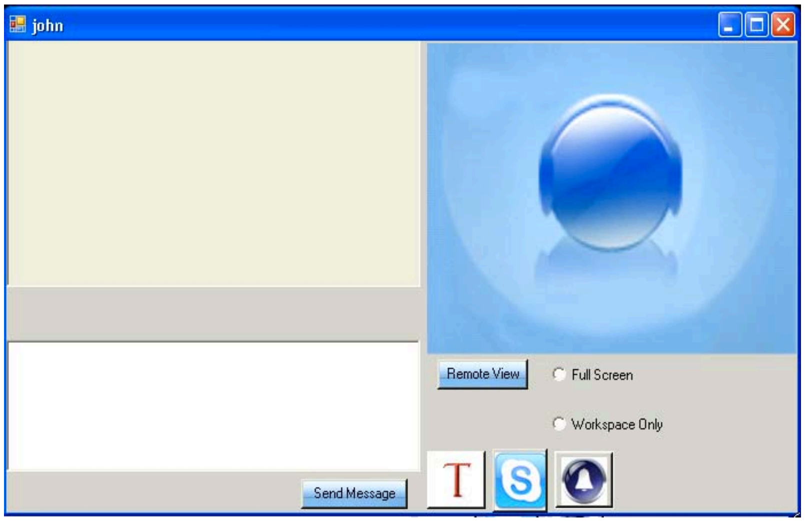


Figure 5-4:NX MUG Layout

Figure 5-6:NX MUG Layout

5.1 NX MUG Demonstration

Three participants collaborated in the demonstration, and spent some time coordinating and decomposing their tasks beforehand. They logged in as John, Lu and April and the group collaborated to design a turbine engine front frame that required multiple features. John started to sketch the inner ring and asked April if she finished the first sketch in XY plane at origin, as shown in Table 5-1 Row A.

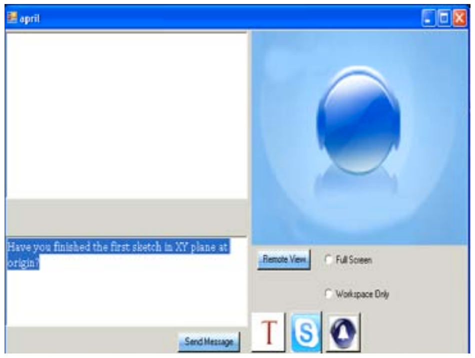
April responded to John that she just finished the sketch and would start a revolve about the x axis a full 360 degrees. April wanted to watch John's screen to make sure there was no overlap of their respective parts in Table 5-1 Row A. John accepted her request and gave April full screen access as shown in Row B. Meanwhile, April was checking other user profiles as shown in Row B. John wanted to watch Lu's workspace and Lu accepted John's request as shown later in Row B and C.

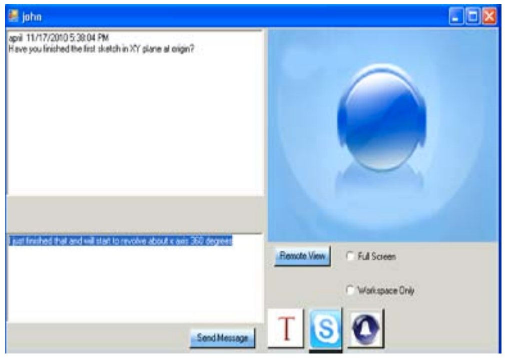
Later April sent John a text to request that she be able to view his workspace. John changed his view setting based on April's request as shown in Row C. Then, April sent a request to watch Lu's screen, but Lu was in the middle of editing the model, and he refused April's request as shown in Row D. Later, April asked Lu about his progress and she typed and sent the message -"hey how is your progress" in her native language (English) to Lu. Lu received the message and translated it into Chinese, Lu's native language, using Google Translator accessed through the MUG; see Rows D and E.

Shortly thereafter, April sent several text messages to Lu and Lu used the Google Translator in Row E. He figured that it might be easier to understand April's intent using the remote view in Row F. Later, April sent an alert to Lu and John and notified them that the model had appeared to be finished as shown in Row F.

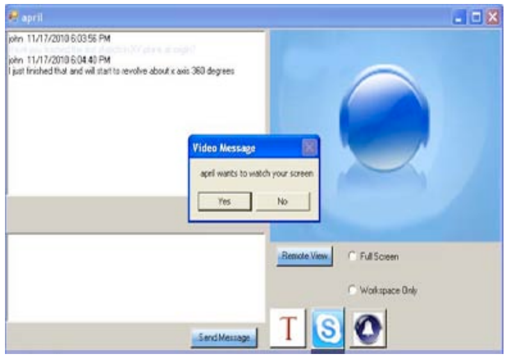
April decided to conclude the session using a video chat rather than texting, and each of them evoked Skype through NX MUG as shown in Row F, Figure 5-7.

Row A


Have you finished the first sketch in XY plane at origin?


april 11/17/2010 5:38:04 PM
Have you finished the first sketch in XY plane at origin?

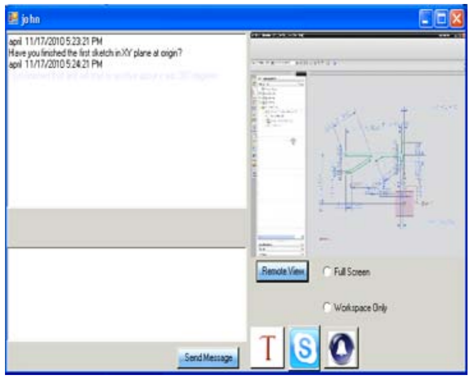
I just finished that and will start to revolve about x axis 360 degrees

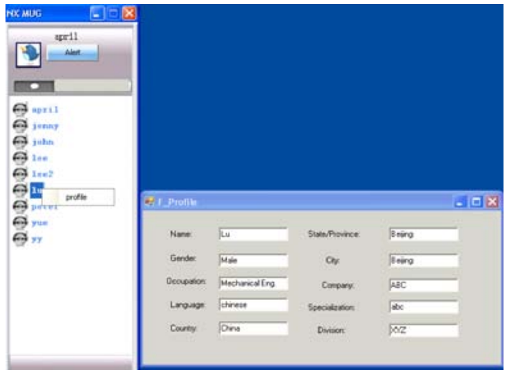

john 11/17/2010 6:03:56 PM

john 11/17/2010 6:04:40 PM
I just finished that and will start to revolve about x axis 360 degrees

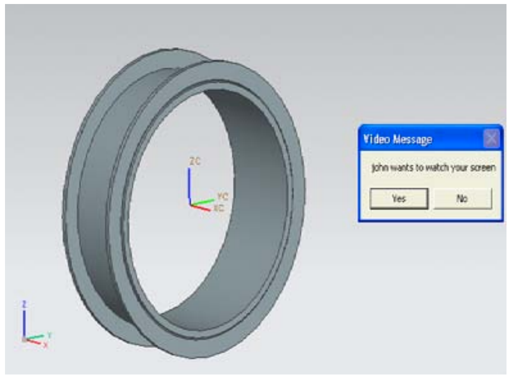
Video Message
april wants to watch your screen

Row B

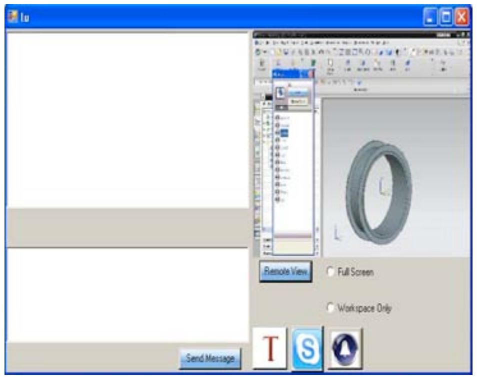

april 11/17/2010 5:23:21 PM
Have you finished the first sketch in XY plane at origin?
april 11/17/2010 5:24:21 PM

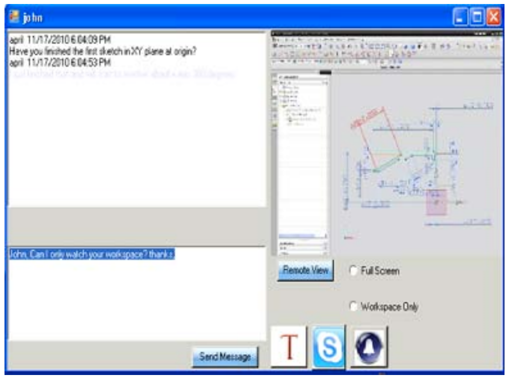


Name:	Lu	State/Province:	Beijing
Gender:	Male	City:	Beijing
Occupation:	Mechanical Eng.	Company:	ABC
Language:	chinese	Specialization:	abc
Country:	China	Division:	XYZ


Video Message
john wants to watch your screen

Row C




april 11/17/2010 6:04:09 PM
Have you finished the first sketch in XY plane at origin?
april 11/17/2010 6:04:53 PM

John, Can I only watch your workspace? thanks

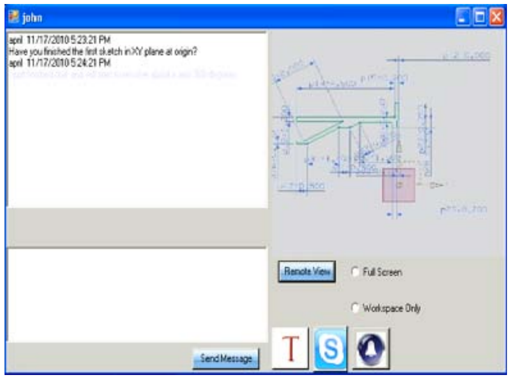

april 11/17/2010 5:23:21 PM
Have you finished the first sketch in XY plane at origin?
april 11/17/2010 5:24:21 PM

Table 5-1: MUG Demo

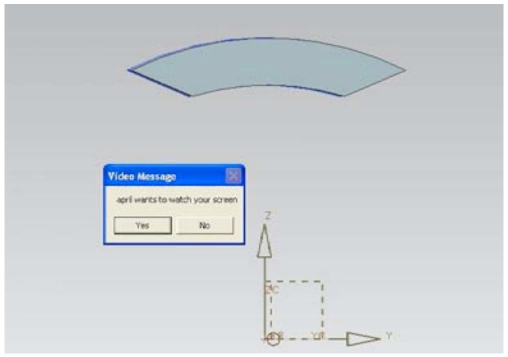
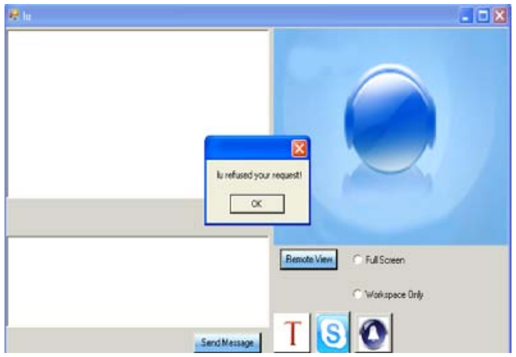
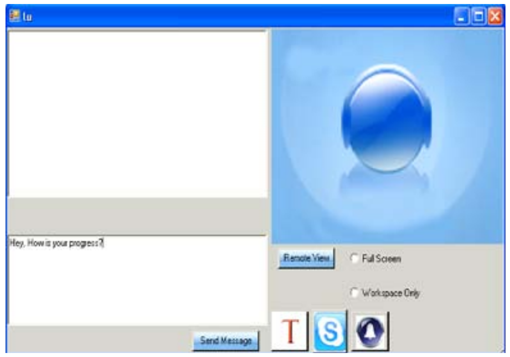
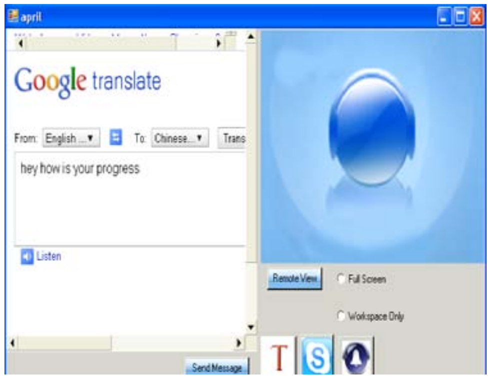
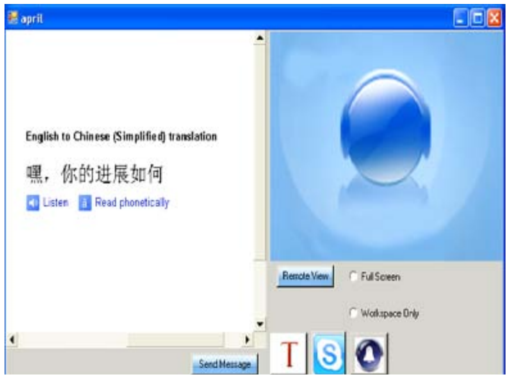
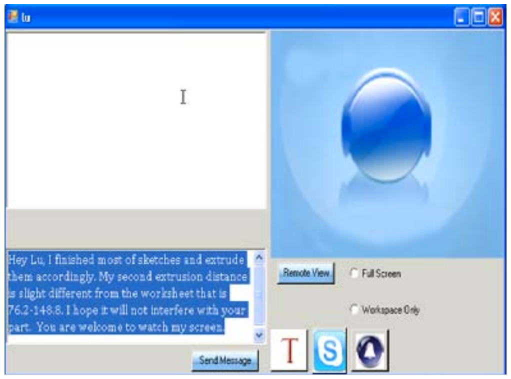
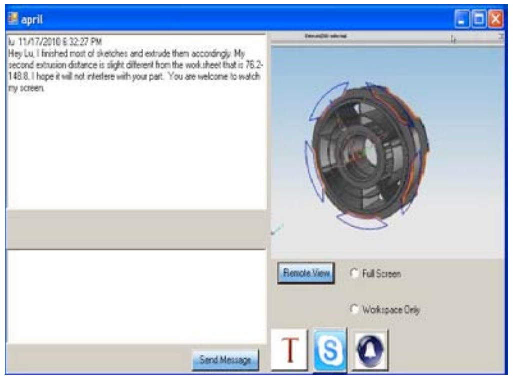
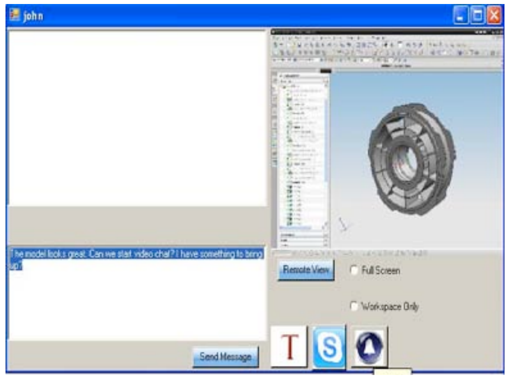
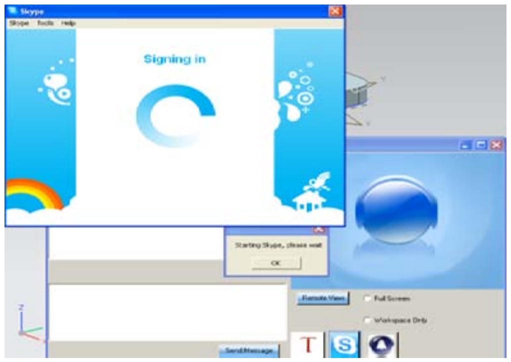
Row D	 Video Message april wants to watch your screen Yes No	 lu refused your request! OK	 Hey, How is your progress?
Row E	 Google translate From: English To: Chinese hey how is your progress	 English to Chinese (Simplified) translation 嘿，你的进展如何 Listen Read phonetically	 Hey Lu, I finished most of sketches and extrude them accordingly. My second extrusion distance is slight different from the worksheet that is 76.2-148.8. I hope it will not interfere with your part. You are welcome to watch my screen.
Row F	 lu 11/17/2010 6:32:27 PM Hey Lu, I finished most of sketches and extrude them accordingly. My second extrusion distance is slight different from the worksheet that is 76.2-148.8. I hope it will not interfere with your part. You are welcome to watch my screen.	 The model looks great. Can we start video chat? I have something to bring up.	 Signing in Starting Skype, please wait OK

Table 5-1: MUG Demo

Figure 5-7: Skype Session

5.2 Ideal MUG Architecture Versus that Currently Achievable

The ideal NX MUG will give each user the right to specify how they interact with other users and will determine what data will be displayed to the users. MUG widgets available through buttons, icons and pop-up panels reside on each computer and let the user choose how to interact with other multi-users. The NX MUG architecture would preserve most current interface features and functionalities of the single-user CAx environment, but extend the single-user environment to allow collaborative/concurrent model creation, editing, deletion, etc.

The future NX MUG should be capable of all the current features, plus those that follow, assuming that future CAx applications will be modified to incorporate these capabilities: 1) integrate some MUS capability or provide API interfaces; 2) provide direct access through CAx API to event buffer, using interrupts to notify when parameter changes occur and allowing API's to copy the data object if necessary. It is also presumed that display communication software like RGS can be modified for multi-user interaction/awareness.

- NX MUG can run as application concurrently with CAx application or will have API set for direct integration into CAx applications, where CAx has some MUS capability.
- NX MUG architecture will allow it to be used in CS, hybrid P2P, or Cloud Serving modes.

- NX MUG will apply security measures to restrict/manage collaboration between users (what models they can view, change, copy, etc.). This will require some interfacing between the NX MUG application and the CAx MUS.
- Future NX MUG can set all the contexts discussed in the thesis, including display control contexts, communication contexts, supervisory control contexts, security contexts, and recording contexts.
- NX MUG integrates auto text translation so that each user communicates in a language preference/context.
- Users set up multi-user display sessions, partitioned on one screen or over multiple displays.

Specifically**,** NX MUG proposes several communication-related functions such as Skype, language translation, radar view, and alert for new updates, but there are some differences between the author's predictions of the original design and the real design.

Ideally, NX MUG would implement automatic sound effects whenever a user updates the model, based on sound alert timing contexts. To avoid being distracted by alerts, a user should be able to set the timing for sound effects updating, whether broadcasting the sound effect or whether receiving a sound update. Another option is that the pop-up window only vibrates in the future when the alert is turned off. But due to the lack of the interrupt in NX, a user can currently submit any changes by pressing a button to broadcast the alert to other clients. It is still useful to have this function when one user makes an important change and wants to get another user's attention.

Also, there are two options of opening the radar views; one option is to view the entire NX window and the other option is to view the workspace only, as shown in Figure 5-8. However, a user can only choose the second option, which is to look at the workspace, after maximizing the NX window. By maximizing the NX window, the application window is sized to the display and the programmer can easily detect the NX workspace size. Otherwise, it is hard for a designer to only capture the workspace region.

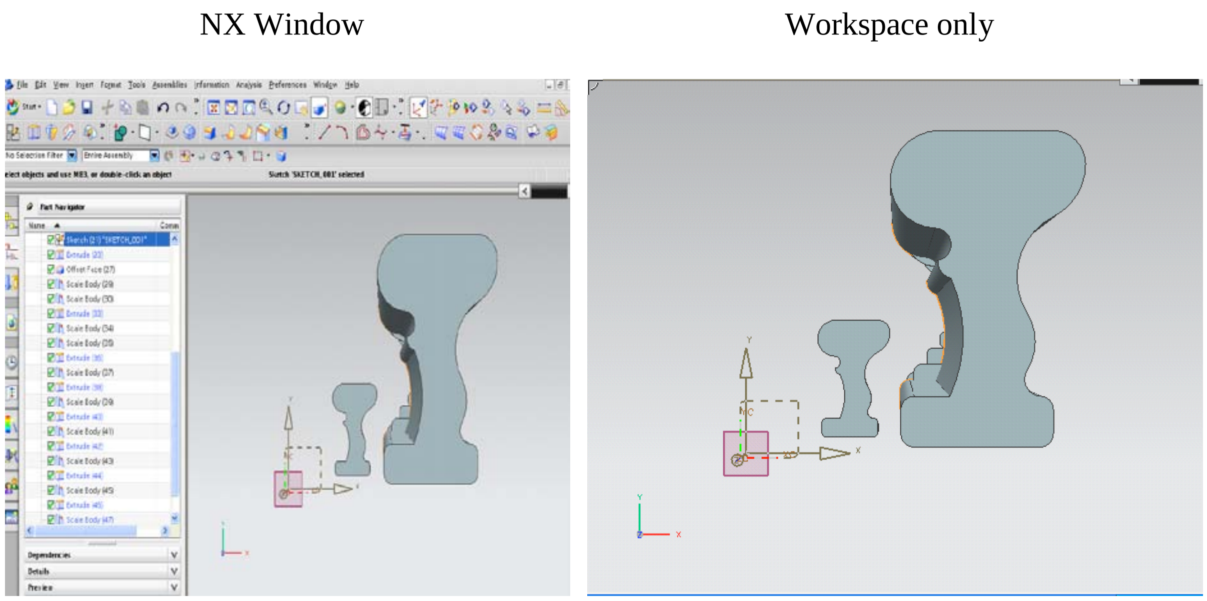


Figure 5-8: NX MUG View Option

In an ideal situation an avatar will be selected before entering the team mode, which is similar to selecting a role before starting a multi-user game. Also a user will change icon based on preference. This action will request a designer to gather a certain amount of avatars, store them in a library and put them in the local folder. When a user installs the MUG, this library will be installed on your computer as well. Due to the complexity of the network option, this option will be considered in the future work.

For log in security the designer currently assumes any user that can access the NX Connect and NX MUG applications can register an account and log in to the application. More intense security will be enabled in the future with special project related passwords, using procedures to assign specific individuals to specific projects. Server methods will be used to observe user activities.

Using server security methods anyone who is able to provide the correct log in and password can register an account and get approved for the assigned project. In this way, the new user first gets authorization from the project supervisor and then participates in the design team. Other protections can be used, such as thumbprint, speech recognition and face recognition over a webcam.

Expectedly, the live chat and text mode will be categorized into private chat/text, which will be limited to the design team audience. Note that in an online gaming environment, users can switch the private/public mode to speak to one person, a group of players or all the players online. NX Skype can potentially be extended to video conferencing in future work.

The ideal NX MUG will be able to partition user displays among multiple displays or divide one display screen, so that a user can see the collaborating design progress among several users; however, the prototype NX MUG is restricted to viewing one multi-user workspace at a time.

The ideal NX MUG will depend on project management software like PLM to support web-based collaboration functions and multi-user accounts, particularly when dealing with complex projects. For example, in a CS architecture, as a user is added to a team or the user changes profile information, the project management software will support the task of synching this information with other clients when they are currently on-line. The current NX MUG prototype uses a registry window. This information is presently stored in the local server and replicated for the client users. The NX MUG prototype is implemented as a CS architecture. The future NX MUG will also work in hybrid P2P and Cloud Serving network architectures.

6 CONCLUSIONS

The objectives of this research are to propose a flexible context architecture for a MUG and demonstrate a simple prototype with current capabilities.

6.1 Research Objective 1

- Investigate and develop architectures for multi-user interfacing of localized and distributed collaborators applying CAx applications, including agent software that can be configured for user context preference.

  A MUG architecture was proposed that includes two major modules: a CAPP module to manage the client MUG interaction with other multi-user MUG's and an ISM module to store and manage multi-user profiles and related context information. The architecture is flexible enough to be deployed over CS, P2P and Cloud Serving networks, and thus is adaptive to resource variation in local or global networks.

  In the proposed MUG architecture, agent software uses context settings to determine what information is to be transmitted between the collaborating users. Again, an issue to be resolved in the future, and which depends somewhat on resources, is whether the agent software strips the data packets locally, depending on context settings, or whether the agent software transmits the context settings to a server to strip the packets before distribution to clients.

6.2 Research Objective 2

- Determine how to integrate filters to provide personalized and selective contextual presentation of collaborating user interactions. A filter provides a function/algorithm to limit collaborating data input, to provide a different view of that data, or to promote multi-user awareness. Filters could be mapped to Multi-User GUI (MUG) menu items or selection icons, etc.

In Section 3.1.1 and 3.1.2 five flexible contexts preferences are proposed: display control contexts, communication contexts, supervisory control contexts, security contexts and recording contexts. The proposed architecture uses simple widgets to set the desired context preferences and agent software is used to transmit the collaborative data between the users according to each user's context preferences.

6.3 Research Objective 3

- Develop MUG prototypes that are consistent in look and feel with current commercial CAx GUI's, but can be invoked or hidden. The intent is to develop software libraries that can be linked to CAx API libraries, and then eventually become a member of the API library after commercialization by the CAx company. In addition, a prioritization method will be considered that allows manager oversight of multi-user sessions.

In Chapter 4 a method was discussed that would allow the MUG to be integrated into a MUS or run parallel with a MUS like NX Connect. NX Connect limitations relate to limitations in Siemens NX API which is designed for single users. Limitations such as lack of event interrupts, no direct access to the parameter change events, geometry handle memory addresses rather than actual geometry parameters, and other limitations make it difficult to integrate any MUS that uses the NX API with a desired MUG. However, a future NX MUG will run as an application layer concurrently with CAx application or have an API set for direct integration into CAx applications, when the CAx supports a MUS capability.

6.4 Research Objective 4

- Demonstrate that MUG's can be practically used to enhance multi-user CAx; and that product personnel of different cultures can navigate CAx collaboration effectively, in spite of cultural differences.

Chapter 5 introduces the basic functionality of the prototype MUG and discusses the current implementation. The MUG prototype provides multi-user currently not available in NX Connect. It also promotes multi-user communication, regardless of where users are physically located, although the prototype was only tested over a LAN.

The implementation in Chapter 5 shows the flexible CAx GUI is feasible**.** This is accomplished by the creation of a custom MUG, including the functions of video, chatting and text translation. The prototype is interfaced with a MUS prototype called NX Connect and uses the Siemens NX, Version 6 CAD application. The NX MUG and MUS application is programmed using the NX CAD API functions and the Visual Studio API to demonstrate all of the desired operations. Specifically, this implementation is able to perform the following operations:

- The user-interface window contains each client's profile, which is consistent with user-registered information**.** The information helps set context preferences in real-time and confirms the 2nd and 3rd objectives of the thesis. For example, the profile information includes which country a user is from and preferred languages; this information can be used in real-time design.

- The main window and pop-up windows can be invoked or hidden at any time.
- A window will pop up when a client clicks on another client icon.
- The pop-up window includes a radar view that shows the specific client's NX workspace, Skype, and Google translator (not limited to Google products**).** These functions reinforce the thesis objectives of promoting multi-user interactions and limiting cultural variations. The radar view can help users cross languages barriers and visually demonstrate model design processes between users. The translator application enables text to be viewed by each user in their language of preference.
- An alert can be sent to a selected individual or the whole team after a part of the model has been changed.

This thesis proposes a flexible MUG architecture that can be applied over various distributed and collaborative architectures, either integrated into CAx applications or concurrent with CAx applications. In conclusion, it is possible to build a flexible MUG used in CAx applications and advocate multi-user awareness.

6.5 Future development

There are several things to consider for future research improvement:

- The investigation of a timer in multi-thread API applied to NX Connect will be done. The reason for using *System.Threading.Timer* instead of *System.Windows.Forms.Timer* is illustrated in 4.3. This investigation may improve the performance of NX Connect.
- The adoption of multiple event buffers and interrupts in CAx applications will enhance the development of MUG and MUS.

- The task of capturing the pictures can be accomplished in a better way, like a third party screen capture tool like RGS, which will use memory effectively.
- Further investigations on display control contexts, communication contexts, supervisor control contexts, security contexts, and recording contexts are suggested.
- Security measures to restrict/manage collaboration between users (what models they can view, change, copy, etc.) will be further investigated. This will require some interfacing between the NX MUG application and the CAx MUS.
- More functions can be added to the current MUG, such as the ability to lock regions, to show multi-cursors in one screen, and to grant permission to edit. These functions need to be accomplished after successfully decoupling the regions. The details about future MUG are also discussed in 3.1.

These improvements will result in improving the performance of NX Connect and creating a more dynamic and robust MUG.

REFERENCES

[1] Liu, Q.; Cui, X.; Hu, X.: *An Agent-based Intelligent CAD Platform for Collaborative Design,* ICI 2008, CCIS 15, pp. 501–508

[2] Shen, Y.; Ong, S.; Nee, A.: *Collaborative Design in 3D Space*, VRCAI 2008, Singapore, Dec. 8 –9

[3] Gelernter, D: *Mirroro Worlds*. Oxford University Press, NewYork, 1993

[4] Martin, D; Cheyer, A;Moran D,:*The open agent architecture:A framework for building distributed software system*, Artificial Intelligence center

[5] Hill, J: *A Direct Manipulation Toolkit for Awareness Support in Groupware*, University of Saskatchewan, August ,2003

[6] Coutaz, J, *Software Architecure Modeling for User Interfaces*, Laboratoire CLIPS(IMAG)

[7] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, R.M.Y. Young, "Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties", *Proceedings of the INTERACT'95 conference*, S. A. Arnesen & D. Gilmore Eds., Chapman&Hall Publ., Lillehammer, Norway, 115-120 ,1995

[8] Gutwin, C, "Workspace Awareness in Real-time Distributed Groupware", Calgary, Alberta, Dec., 1997

[9] Wallace, G;Bi,P;Li,K.;*Anshus,O:A Multi-cursor X Window Manager Supporting Control Room*, http://www.cs.princeton.edu/omnimedia/papers/multicursor.pdf

[10] Stødle, D: *User Interface Components to Support Simple and Efficient Use and Control of Large*, high resolution displays, University of Tromso, 2005

[11] Bahill, T*: Applications of Blackboard Architecture, systems and industrial engineering*, University of Arizon, 1997

[12] Red, E; Jensen, C; Holyoak, V; Marshall, F; Xu, Y: "v-Cax:A Research Agenda for Collaborative," *Computer-Aided & Applications*, Vol. 7, No. 3, pp. 387-404, 2010

[13] Ryskamp, J; Jensen, C; Mix, K; Red, E: Leveraging Design Rationale to Improve Collaboration in Multi-user CAD", Accepted for publication in Tools and Methods for Competitive Engineering, 2010

[14] Gutwin, C.:“Traces: Visualization of Interaction.” *Department of Computer Science*, University of Saskatchewan, Saskatoon, SK, Canada, 2001

[15] Douglas, S.; Tanin, E.; Harwood, A.; Karunasekera, S.: *Enabling Massively Multi-player Online Gaming Applications on a P2P architecture*, Proceedings of the International Conference on Informa- tion and Automation, December 15-18, 2005, Colombo, Sri Lanka, 7-12

[16] Eiji, A:*Knowledge and Skill Chains in Engineering and Manufacturing Information infrastructure in the Era of Global Communications*, Proceedings of the IFIP TC5 / WG5.3, WG5.7, WG5.12 Fifth International Working Conference of Information Infrastructure Systems for Manufacturing 2002 (DIIDM2002), November 18–20, 2002 in Osaka.

[17] Sosa, M., Eppinger, Rowles, C., “A Network Approach to Define Modularity of Components in Complex Products,” *ASME Journal of Mechanical Design*, 129 (11), 1118-1129, 2007

[18] McFarlane, R.D.P., "Network Software Architectures for Real-Time Massively-Multiplayer Online Games," M.S. Thesis, *McGill University, School of Computer Science*, Feb., 2005

[19] Samar Ammar-Khodja, Nicolas Perry, and Alain Bernard, “Processing Knowledge to Support Knowledge-based Engineering Systems Specification,” *Concurrent Engineering*, 16, 89-101, 2008

[20] Comaya Official Website, http://cooffice.ntu.edu.sg/comaya/video.htm

[21] User Interface Design Tips, Techniques, and Principles,Amysoft, http://www.ambysoft.com/essays/userInterfaceDesign.html

[22] Science Direct site, http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6TYR-4D9D8W63&_user=456938&_coverDate=04%2F15%2F2005&_rdoc=1&_fmt=full&_orig=search&_cdi=5625&_sort=d&_docanchor=&view=c&_searchStrId=1144551325&_rrigin=google&_acct=C000021830&_version=1&_urlVersion=0&_userid=456938&md5=c5d35d60a46e9029e3c6d6a0513d4927#SECX16

[23] Fuh, JYH;Li, W.D.: “Advances in collaborative CAD: the-state-of-the art,” *Computer-Aided Design, ElSEVIER*, 2004

[24] Schollmeier, R: *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE, 2002

[25] Wikipedia, http://en.wikipedia.org/wiki/Architecture#References

[26] Bentley, R., Roden, T., Sawyer, P, Sommerville: *Architectural Support for Cooperative Multi-user Interfaces*, Computing Department, Lancaster University , 1994

[27] Lauwers, J. C., Lantz, K. A., *Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window Systems*, Proceedings of CHI '90, April 1-5, 1990, Seattle, Washington, ACM Press, pp 303-311

[28] Zhou, H.; Han, M: “CAD/CAM Optomechatronics, in Education and Training in Optics and Photonics”, *OSA Technical Digest Series* (Optical Society of America, 2003), paper EWF7

[29] Kao, Y-C; Lin,G: Development of a Collaborative CAD/CAM System, *Robotic and Computer-Integrated Manufacturing14*, 55-68, 1998

[30] Larson, B.M: “Exploring the Common Design Space of Dissimilar Assembly Parameterizations for Interdisciplinary Design”, *Dept. of Mechanical Engineering, Brigham Young University*, August 2008

[31] Jo, C; Chen, G;Choi, J: *A framework for BDI Agent-based Software Engineering*, Studia Informatica Universalis

[32] Wikipedia, http://en.wikipedia.org/wiki/Polling_%28computer_science%29

[33] Eck, D: Introduction to Programming Using
va, ***http://math.hws.edu/javanotes/c1/s2.html***, 2006

[34] Hofstede G: *Cultures and Organizations: Software of the Mind*, New York: McGraw-Hill U.S.A. 2003

[35] Technology & The Internet by Andy Finn:http://www.tafinn.com/andyfinn-us/Writing/Technology/emoticons.htm

[36] Wikipedia, http://en.wikipedia.org/wiki/Cloud_computing