49

Distributed Software Design for Collaborative Learning System Over the Internet

by

Christine Hui Su

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering in Electrical Engineering and Computer Science and Bachelor of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998 [illegible]

Author ....
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by.....................
Feniosky Peña-Mora
Assistant Professor
Thesis Supervisor

Eng

Accepted by.........
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Distributed Software Design for Collaborative Learning System Over the Internet

by

Christine Hui Su

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 1998, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Electrical Engineering and Computer Science and Bachelor of Science in Electrical Engineering

Abstract

In this thesis, a software design approach is introduced to solve the distance collaborative learning system problem. As Internet usage becomes popular, people from all disciplines seek to utilize the net as both a communication and learning tool to develop projects and improve work efficiencies. However, there are limited collaborative tools available in the market, and the research in distance communication has not tackle the problem of providing casual contact along with social interaction features. This thesis presents an applicable object-oriented design for distance collaborative system using Unified Modeling Language notations. In addition, the benefits and pitfalls in working with a diversified and geographically separated team is shared in this thesis.

Thesis Supervisor: Feniosky Peña-Mora
Title: Assistant Professor

Acknowledgments

First of all, I would like to thank Professor Peña-Mora for introducing me to this interesting project and giving me insightful guidance through out the thesis phase. I would also like to extend my appreciation to my CICESE and MIT instructors and team members, Jesus Favela, Josephina Rodriguez, Bob Yang, Felix Loera, Simonetta Rodriguez, Humberto Chavez, Rene Navarro, Sergio Infante, Kareem Benjamin, Juan Contreras, Gregorio Cruz, Lidia Gomez, Diana Ruiz, Charles Nijendu, Juan Fancisco, and Marcela Rodriguez. Without them, the project would not be completed. In addition, I would like to acknowledge the teach assistants, Karim Hussein and Siva Dirisala, who had spent time in setting up equipment and videotyping our progress.

In my five years at MIT, I also had gotten to know a few people who had became good friends of mine. I would like to thank them for being with me through the good and bad times, tolerated my bad behaviors, listened to my complaints and shared my happiness. Countless thanks to Cindy, who had made my MIT life a lot more interesting; to Sanjeev, who has been more than a great boss; to Inn, who is a God sent angel and, who introduced me to Christianity; to Henry, who I have gotten to know better lately, it was wonderful to have him as a friend; to Marcus, who had taught me so much about life outside of MIT; and to Bob, who is talented and wonderful to work with.

Least not the less, I would like to thank my parents and my sister. My mom is the most incredible person I have known, and she continues to be my best friend throughout the past five years. I just cannot say enough thanks to express my gratitude towards her. I also want to thank my little sister, who is intelligent, energetic, responsible, and creative. Finally, I would like to extend my appreciation to Jinzy Zhu, Patrick Yeung, Yoko Kusumoto, Sunnia Lin, and Jay Ongg. They had been my friends both in present and past, and they will not be forgotten.

Contents

List of Figures

[1]15.564, Information Technology I is a course from MIT Sloan School of Management, taught by Professor Chrysanthos Dellarocas.

List of Tables

Chapter 1

Introduction

This thesis is developed from the studies in the Distributed Software Engineering Laboratory (DISEL). In this laboratory, students from Centro de Investigación Cienífica y de Educación Superior de Ensenada (CICESE) and Massachusetts Institute of Technology (MIT) collaborate in the development of a medium scale software project. Each participant plays a role within the software development cycle of the project.

This laboratory proposes a new perspective of distance education by having students experience a professional product development process. At the same time, students would focus on the collaborative nature of his/her learning experience while working with each other over distance to gain further insight to the existing problems within collaborative learning. The class is modeled in resemblance to a real software company, where each student will be involved in one or more specific roles, such as Project Manger, Analyst, Software Designer, Programmer, Quality Control Engineer, Validation and Verification Engineer, Maintenance Engineer and Documentation Specialist.

To promote the entrepreneur spirit, the team's project proposal was entered to the MIT's Entrepreneurship 1K Competition. The feedback from the judges in the competition were positive and encouraging, showing the possibilities of a business application of the project.

This chapter will serve as an introduction to the domain of work in distributed system design and collaborative learning. The motivation of solving some commonly known social and casual interactions problem in distant communication will be presented. The remainder of this thesis is divided into six chapters, each covering a piece in defining and solving the distance communication problem.

In Chapter two, a general overview of three fundamental concepts will be introduced. These concepts include collaborative learning and software engineering. The definition of software design and its purpose within software engineering will also be discussed in depth.

Chapter three describes the background of the project, the motivation of developing a software to solve the distance education and collaborative learning problem. A brief coverage of the Unified Modeling Language (UML) notation and conventional Object-Oriented Design will be provided. A section of this chapter will also be devoted to explain the role of design, its related theories, methodologies, and processes that were used in this research.

Chapter four examines some current existing distance communication tools, such as the popular NetMeeting and the game-like 3D World Chat. Comparisons of the DISEL system with each of these tools will also be given in this section.

Chapter five covers the requirement analysis for the DISEL product. The work was mainly conducted by the analysts of the team, Semonetta Rodriguez and Humberto Chavez. The presentation of requirement analysis can give reader a sense of what is to achieve through this project. It is also a yard stick for performance and quality measurement of the design and implementation.

Chapter six proceeds to the core of the DISEL system design, details of an extensive collection of documents will be presented. Those include the requirement analysis for the research, design document, system feature document, system architecture model, object model, and quantitative specification.

Chapter seven evaluates the quality of the DISEL design. This evaluation comes from both a technical and interpersonal point of view. It will provide some insights of distance development difficulties, not to mention, some very rewarding experience as well. This chapter also draws a conclusion of the distributed software design for collaborative learning. Issues of feasibilities and future improvements will be discussed.

1.1 Importance of Distance Learning

The terms "distance education" or "distance collaborative learning" have been applied interchangeably by many different researchers to a great variety of programs, providers, audiences, and media. Its hallmarks are the separation of teacher and learner in space or time, the willful control by the student in learning rather than the distant instructor, and noncontiguous communication between student and teacher, mediated by text, graphics, audio or video.

Until recently, the use of traditional non-interactive or transmissive media in distance education only provided trainers with the option of pedagogic methods in which individual learning is predominant. Learners in distant geographical locations were left isolated and, thus, deprived of learning methods that originated from interactive communication in a close social setting. Hence, there is a desperate need in developing system that will enable social or cognitive interactions.

Collaborative learning, generally considered to be a method reserved exclusively for face-to-face situations, is now possible to implement through distance education. Many view the popular video conferencing as a successful approach to distance learning. However, if one examine the concept of social interaction closely, using video conferencing to communicate is greatly flawed. Social interaction is constituted of three elements: social feedback, casual contact, and personal expression. It is clear that, the first and last of the these are directly related to each other, and they are crucial for establishing lasting bond between connected parties. The signals, such as facial expression, speech tone, and hand movement can proved to be very valuable in delivering intentions during conversation. An ideal learning system will provide an environment that will allow uninhibited social interactions and free information sharing. The Figure 1-1 below expresses the concepts of social interaction in context of collaborative distance learning.

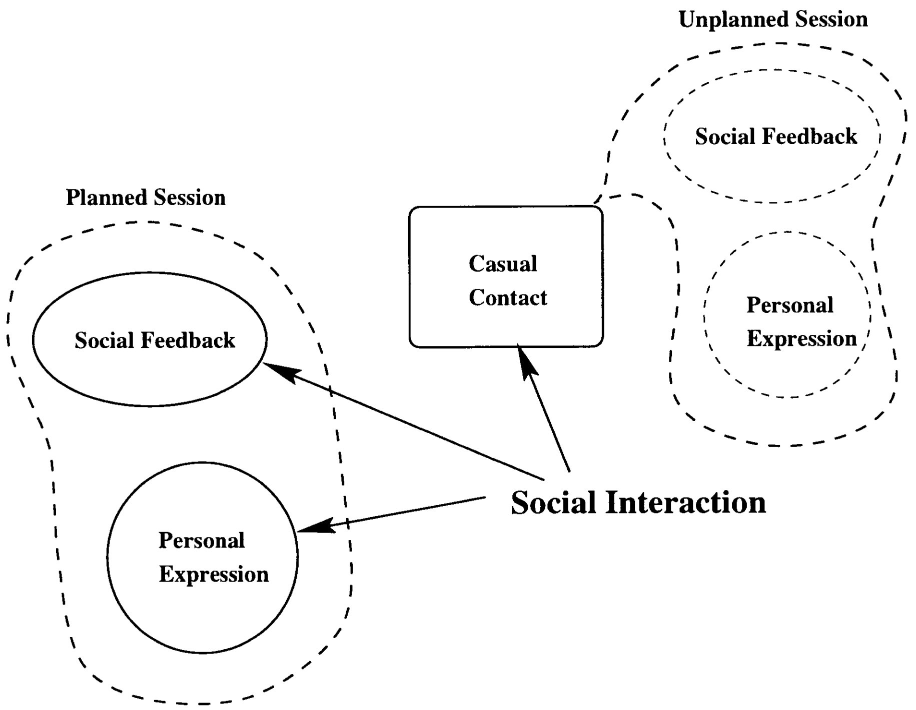


Figure 1-1: Social Interaction

The distance learning problem does not just attract academic attention, policy makers and the general public are also increasingly interested in solving this problem. Once it is solved, the solution will be an extremely valuable option for communications between remote areas. Distance learning allows users to hear or see the instructor's teaching, as well as allowing teacher to react to his audience's comments and questions. Moreover, virtual learning communities are not bounded by locations, and they can be as close as co-located learning groups.

The scope of distance learning problem is broad. However, lying inside the core of the problem is how to create and maintain trusting and lasting bonds between people. How do one create an virtual environment that will enable distant social interaction seemed local? The question is challenging, because there is no definite answer for it. One has to engage himself to understand the problem, define the problem and then proceed to solve the problem. This thesis will provide some insights and techniques for such a process.

1.2 Objective

The distance learning software application developed by the DISEL group is aimed at providing a solution to the problem. The current techniques and technologies for conducting distance learning and distributed project collaboration contains significant deficiencies. There is severely limited or non-existing social interaction between participants in these applications. For example, the chat room environment that exists on the web is largely used to communicate information, but users of these chat rooms do not view the connection as a representation of their truthful thoughts. Many times these users use false identities to conceal themselves, and the textual information they exchanged serve more as an entertainment purpose rather than an educative one.

The goal of the project is to enforce a meaningful and realizable social interaction between distant parties that are using the DISEL system. Unlike the chat rooms, it is intended to provide educational services. The system will connect people across data network, and allow them to engage in conversations, meetings, lectures, and discussions. The virtual environment implemented will give individuals freedom to express themselves without the constraints of the machine. In addition, users can experience virtual classroom or meeting situations through electronic means, that will not deviate greatly from the real ones.

1.3 Benefits

Major effort of the system will be devoted to provide casual interaction services to the users. There are a lot of existing implementations on the market that have partial distant planned interaction solutions. For example, the Microsoft Netmeeting application can transmit video and audio across the Internet, and it also allows users to share graphics. However, all conversations conducted in NetMeeting are planned. Users are prepared to engage in such interaction. In Contrast, a casual interaction is not planned. It is a form of communication where users meet with someone unintentionally in a virtual setting. The possibilities to have an unstructured conversation between users are greatly beneficial. It will bring the individual distance learning experience one step closer to the real thing.

One important element of distance learning research is about human thinking. It is an integral part of the distance education. Clearly, it is essential to consider human factors thoroughly in a complex system that will provide casual and planned interactions. Though this thesis will not discussed distance learning in psychological perspectives, nevertheless, the study of human behavior in this type of communications is necessary. Most importantly, distance learning is not an isolated phenomenon, it is affected by the political, social, financial and technological factors in its environment. As a result, the understanding of the influence and scope of distance education will benefit many people.

In the next chapter, the discussion will shift to focus on some practical distributed system concepts and system methodologies to solve the collaborative learning problem presented here.

Chapter 2

Concepts, Methodologies, and Theories

2.1 Overview of Software Engineering

Software Engineering is a discipline for software development, it is a combination of using comprehensive methods in each developing phase, and better tools for automating these methods. In short, software engineering provides powerful building blocks for implementation, and good techniques for software quality assurance, work coordination, time or resources control [28].

2.1.1 Definition of Software Engineering

The definition of software engineering can be boiled down to one sentence:

> The establishment and use of sound engineering principles in order to obtain software that is reliable and works efficient on real machine [22].

Software engineering contains a set of three key elements - methods, tools and procedures. These key elements enable the manager to control the process of software development. In the following section, we will briefly examine each of them.

Methods

Software engineering methods provide the technical *how to* for building software. Methods involve a broad array of tasks that include project planning and estimation, system and software requirement analysis, design of data structure, program architecture, and algorithm procedure coding, testing and maintenance. Methods for software engineering often introduce a special language-oriented or graphical notation and introduce a set of criteria for software quality [2].

Tools

Software engineering tools provide automated or semiautomated support for methods. Today, tools exits to support each of the methods noted above. A system for the support of software development, called computer-aided software engineering (CASE) is established to streamline this process. CASE combines software, hardware, and a software engineering database to create a software engineering environment that is analogous to the popular computer-aided design (CAD) system for hardware [11].

Procedures

Software engineering procedures are the glue that holds the methods and tools together and they enable rational and timely development of computer software. Procedures define the sequence in which methods will be applied, the deliverables that are required, the controls that help to ensure quality and coordinate change, and the milestones that enable manager to access progress [24].

Altogether, these three steps are called the software engineering paradigms. Many times, a paradigm is chosen based on the nature of the project and application, the method and tools to be used, and the controls and deliverables that are required. There are three widely discussed paradigm, and a short description of each is provided in the following sections.

2.1.2 Classic Software Life Cycle

Each software development cycle goes through a number of stages, these stages are stepping stones for others. They are also good indicators of the development process. These stages include: system engineering analysis, software requirement analysis, design, coding, testing, and maintenance.

System Engineering Analysis: The process of establishing requirements for all system elements and then allocating some subset of these requirements to software. This step is necessary only if software is a module of the entire system.

Software Requirement Analysis: The process of collecting crucial information about the product from the customers. Requirements define what need to be implemented.

Design: The step that translates requirements to a representation that software can be assessed on. It defines the four distinct attributes of the program: data structure, software architecture, procedural detail, and interface characterization.

Coding: The step of translating design to machine understandable format.

Testing: The process that ensure the software functions perform correctly, and reliably.

Maintenance: An on going process to enhance existing features of the software, and adapt external changes for smooth integrations.

In practice, not all the stated steps are necessary, and not all of them have to follow the restricted order given above. Here, three classic software development processes will be presented, and each would have its own advantages over others in certain situation.

Waterfall Model

The waterfall mode illustrates a systematic sequential approach to software development. The process begins at the system level and progresses through analysis, design, coding, testing, and maintenance. This model is simple and straight forward for actual practice, however it is not without flaws. Many times, iterations are necessary in real-life projects, and this model's structure is too rigid to reflect these type of activities. In addition, the mode requires completion of each step before going to the next stage, increasing the cost of later modification. Although there are problems with the waterfall model, it is still the most basic and easiest to understand template among the three paradigms that will be discussed here [24, pages23–36]. An illustration of the waterfall model is provided in Figure 2-1.

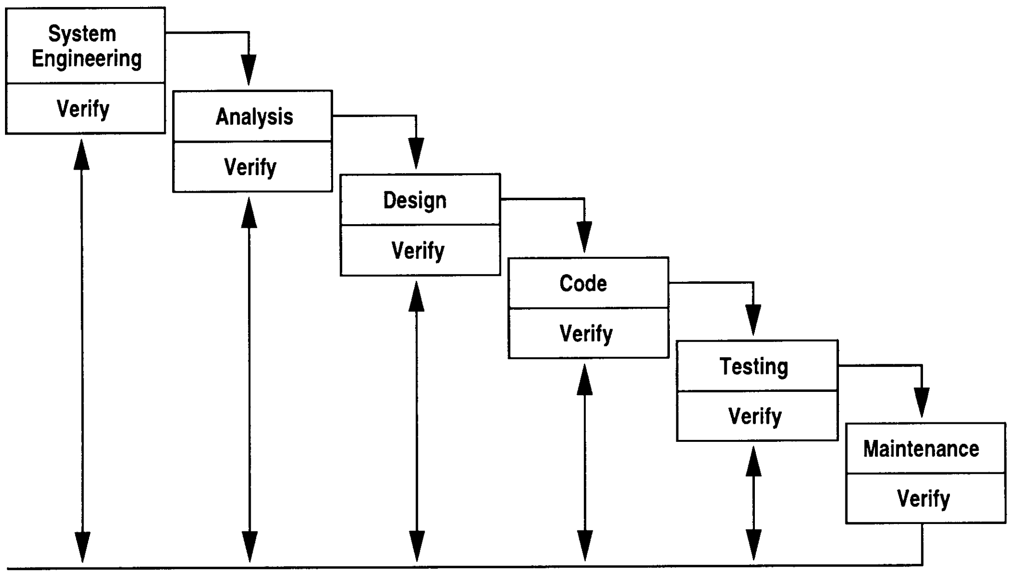


Figure 2-1: Waterfall Model Diagram, Modified from [24, page 226].

Prototyping

Prototyping is a process that enables the developers to create a model of the software that must be built. It offers a better approach to situations with general objectives but no detailed requirements [7]. The model helps developers to perceive the possibilities to realize the feasibility of refining the prototype. The sequence of events for the prototyping paradigm is shown in Figure 2-2. Like all approaches to software development, prototyping begins with requirement gathering, then a "quick design" occur. The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and is used to refine requirements for later version of the same piece of software. A process of iteration continues until the prototype is tuned to satisfy all the user's needs.

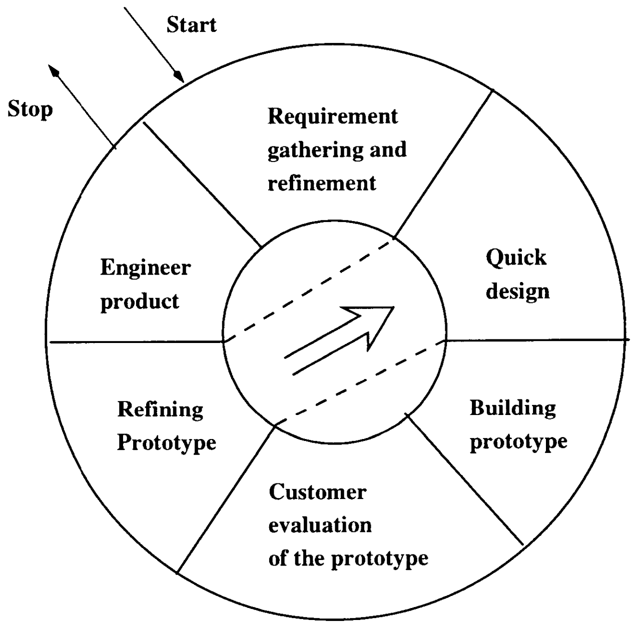


Figure 2-2: Prototyping Model Diagram, Adapted from [24, page 228].

Spiral Model

The spiral model is developed to use the best features of the waterfall and prototyping models [3, pages 61–72]. In addition, the risk analysis is added to the model. The process illustrated in Figure 2-3 is represented by four major activities:

Planning - determination of objectives, alternatives, and constraints.

Risk analysis - analysis of alternatives and identification/resolution of risks.

Engineering - development of the next-level product.

Customer evaluation - assessment of the results of engineering.

The model provides a clear picture of developing reliable software while keeping the cost in check. As each iteration around the spiral completes, the software progresses gradually. If initial prototype reveals major problems of the product, the project can terminate right away, and the risk of complete failure at the end is minimized [14].

2.1.3 Generic View of Software Engineering

The software development process contains three generic phases regardless of the software engineering paradigms that is chose [24]. The three phases: definition, development, and maintenance are encountered in all software disciplines despite different application areas, project sizes, complexities, or resources. Therefore the overview of these three phases is important, it will give us an universal view of the characteristics of software development.

Phase one: Definition

The definition phase focuses on *what*. That is during definition, the software developer will attempt to identify what information is to be collected, what functions and performance are desired, what interfaces are to be established, what design constraints

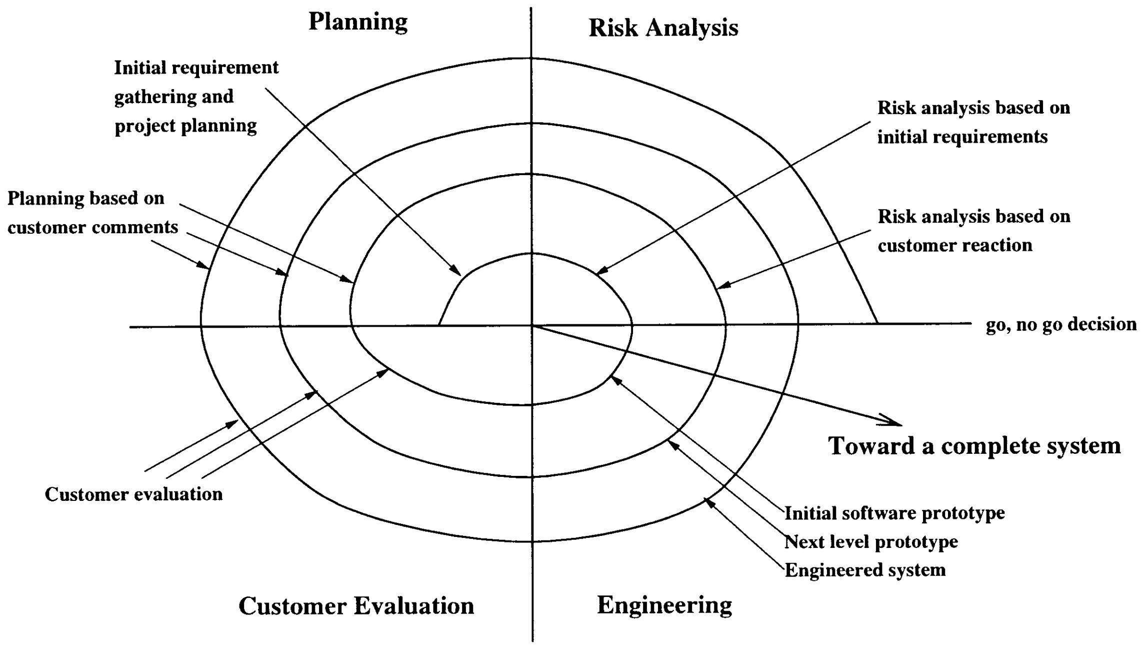

Planning
Risk Analysis
Initial requirement gathering and project planning
Risk analysis based on initial requirements
Planning based on customer comments
Risk analysis based on customer reaction
go, no go decision
Toward a complete system
Customer evaluation
Initial software prototype
Next level prototype
Engineered system
Customer Evaluation
Engineering

Figure 2-3: Spiral Model Diagram, Adapted from [24, page 232].

exist, and what validation criteria are required. The key requirements of the system and the software are identified. Altogether, three steps will occur in this phase, system analysis, software project planning, and requirement analysis.

Phase two: Development

The development phase focuses on *how*. That is, during development, the developer will attempt to define how data structure and software architecture are to be designed, how procedural details are to be implemented, how the design will be translated into a programming language, and how testing will be performed. The methods applies may vary, but the three specific steps are unchanged: software design, coding and testing.

Phase three: Maintenance

The maintenance phase focuses on *change*. Associated with error correction, adaptation required as the software environment evolves and enhancement brought about by changing customer requirements. The maintenance phase reapplies the same steps of the definition and development phases, but does so in the context of exiting software. The three types of changes that happen in maintenance are:

1. Correction - bug tracking and defect repair.
2. Adaptation - modification to software to accommodate changes in its external environment.
3. Enhancement - adding functions/features that are beyond the original requirement.

The importance of these concepts, especially those that link to the role of designer will be reiterate in Chapter 3, where object-oriented design framework is presented. After introducing the definition of software engineering, discussions of Collaborative Learning will be presented in the following section. Since the goal of the project is to design a workable collaborative learning system, defining the meaning of collaborative learning is essential.

2.2 Collaborative Learning

Collaborative learning is an activity described as group attempting to work together toward a common goal [15, pages 167-178]. Though this statement may not be accurate for current on-line experience, it does cover a wide range of possibilities of different collaborative efforts through distance communication.

2.2.1 Definition

Collaborative learning as an intentional teaching and learning strategy has seen tremendous growth in the past twenty years. It has been formally developed, studied, and evaluated in a wide range of spectrum. The technique of collaborative learning is separate into five elements: positive interdependence, face-to-face interaction, individual accountability, interpersonal skills, and group processing [27, pages 137-146]. The study in this thesis remains closely related to the first two elements.

1. **Positive interdependence**

   Fostering a positive interdependence among group participants is very important to a successful collaborative learning experience. Group member need to feel they need each other to accomplish the task at hand. This can be achieved through establishing goals mutually conceived, negotiated and agreed upon by the group. Establish joint rewards to be received by all members is another aspect helps to create an environment of positive interdependence. Shared information and materials are equally important to the group in so far as this provides a basis for members to have insight into the overall task and to e assistance to each other. The final aspect contributing to positive interdependence is the assignment of individual roles. Each person must understand the part they play and their relationship to the whole project. The presences of these aspects are important to insuring a successful collaborative learning experience.

2. **Face-to-face communication**

   As traditionally perceived, collaborative learning involves face-to-face communication. Verbal interactions taking place among students and lecturers of the experience are important. This is a process that incur the most substantial transformation when moving to the computer-mediated environment. Interaction is broken down to separate textual and visual cues, where it is up to the individual on both side to assemble the elements back together to construct a realistic image. Figure 2-4 illustrates this process.

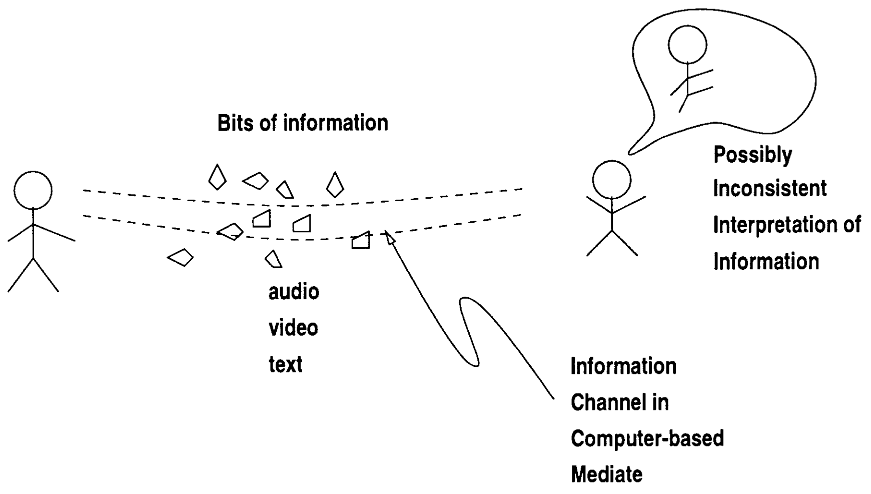


Figure 2-4: Computer Mediated Face-to-Face Interaction

2.2.2 Problems with Distance Collaborative Learning System

As it is described above, the two most important elements in successful collaborative learning is the *positive interdependence* experience shared among the group members and the *face-to-face communication* between participants. However both requirements need members to physically co-located in one place. The difficulty of replicating the traditional collaborative learning experience through electronic media is how to implement an effective interface. The information one wants to transmit is translated to machine understood data to deliver to the distant party. The interface between machine and human is crucial in correctly interpreted the human reaction and filter out the non essential elements. A human can be considered as a black-box which transmits different signals to the other world. How those signals can be processed is up to each individual that receives them. The task of the interface is not to filter or interpret those signals, but to deliver them without modifications.

There are no good ways of building such machine-human interface. The closest replication of the collaborative experience is video conferencing. Though such system is capable of delivering the visual and audible information, it is not capable of delivering subtle mood changes of users. Evidently, the need to have machine and human interaction also creates a barrier between distant parties. Individuals may feel reluctant to voice opinion to people on the other side and visa versa. These barriers are removable through better understanding of the interface between human and machine, and through careful construction of better learning implementations.

Chapter 3

Research Background

The objective of DISEL is to build a system that will improve the existing distance learning system, such as the video conferencing and on-line chatting. None of these systems provide means for users to form lasting bonds that are based on collaborations. Instead of stressing on replicating face-to-face conversational experience, the system we develop will focus on two aspects of distance communications: social and casual interactions. Social interaction possesses standard parameters that are similar to those of a meeting environment. For example, the conversation, the body gesture, the facial expression, and other signals of people that will be used in a typical meeting.

On the other hand, there is the notion of casual interaction, the not so formal encounter between persons. How do we differentiate the nature of social and casual interaction? The simplest concept is that casual interaction involves no intention and expectation. For example, if you walk into a local supermarket, you don't expect to see your professor there, however, if you do meet him, you will try to strike a conversation that is within context. The effect of this context related conversation is very important, you might find out information that you won't get from class. These bits of information may not help you in understand the lecture better, but it certainly is an eye opener for you. It's guarantee that you will view your professor differently the next time in class, because now you know him as a person. Such knowledge is beneficial for human bonding, it is brought forward by casual interactions.

In another example, say you meet with a colleague at the water cooler, who is working on the same project with you. You start a common "Hi-Bye" conversation, and it smoothly diverge to a discussion of your current progress on the project. These types of conversation came about without initial planning, but they do serve purposes in improving communications among team members. I shall classified these category of interactions as the bathroom, coffee machine or water cooler phenomenon.

If casual interaction is important, why is it missing from all the products that are on the market now? One simple answer is that, casual interaction is hard to replicate. If the event does not happen randomly, it will just become another subset of social interaction. Securing the randomness of such event is difficult to achieve, and there is no better approach known so far. Therefore, there exist vast opportunities in the area of casual interaction through distance communications. We believe this enhancement will greatly affect how people communicate and behave on line. In return, such feature will help pushing technology one step further in making virtual experience real.

3.1 Research Motivation

The join effort in experimenting the collaborative distance learning between MIT and CISECE is the main driving force behind this project. Each student will take on a specific role in the software development cycle. In this report, materials will focus on the role of designer for this distance learning system.

Though the need of developing an useful application is the first priority for the project, the purpose of learning through out the process is high up in our agenda. A successful completion of DISEL product may be important, however, the positive experience of each individual can take home after the class is even more important. The true motivation of engaging a group of students in the distance learning project is to help identify the problem in current system, and to come up with new solutions.

The ability of implementing those solutions are minor compare to the abilities to look into the right area, and ask the right questions that will induce imagination and creativity.

3.2 Research Description

The research team is formed with entrepreneur spirits. In fact, the group of students are viewed as members in a software company. The professors server the role of clients who came to the team for solutions to their technical problems. To be consistent with the terms that were used before, the team will be called *DISEL team*, and the product that is developed is called the *Cliq! system.*

The following information is based on a collection of documents that are produced from the project. You should be able to gain a good perspective of the purpose and goal of the project from the materials provided below. Also keep in mind that the specialty of my participation in this research is software design. The majority of materials presented in this chapter will be very much design related.

3.2.1 Team Vision

We are living a revolution whose motor is the convergence between computing and communications. This group proposes to assume a leading role, making technological contributions that impel this revolution. In particular through the successfully developing of a software system that fulfills the specified requirements of functionality, cost and calendar, using modernized human and technological resources, both of them geographically distributed.

3.2.2 Team Mission

Our mission is carrying out us in a professional way, committed with the vision of the group. Each member will assume the assigned role with responsibility and high level of communication and motivation that propitiates a pleasant atmosphere of work. We will express our doubts and problems freely and will be willing to listen and help.

3.2.3 Participants

The team is composed of fifteen members. Six students from MIT, and nine students from CICESE. The division of the role is based on both personal preferences and the logical workload distribution. The organization chart (Figure 3-1) will illustrate this role division relationship.

3.2.4 Role of Designer

In every engineering discipline, design encompasses the disciplined approach we use to invent a solution for some problem, thus providing a path from requirements to implementation. In software engineering the purpose of design is to construct a system that meets the following issues:

- Satisfies a given functional specification.
- Conforms to limitations of target medium.
- Meets implicit or explicit requirements on performance and resource usage.
- Satisfies implicit or explicit design criteria on the form of the artifact.
- Satisfies restrictions on the design process itself, such as its length or cost, or the tools available for doing design.

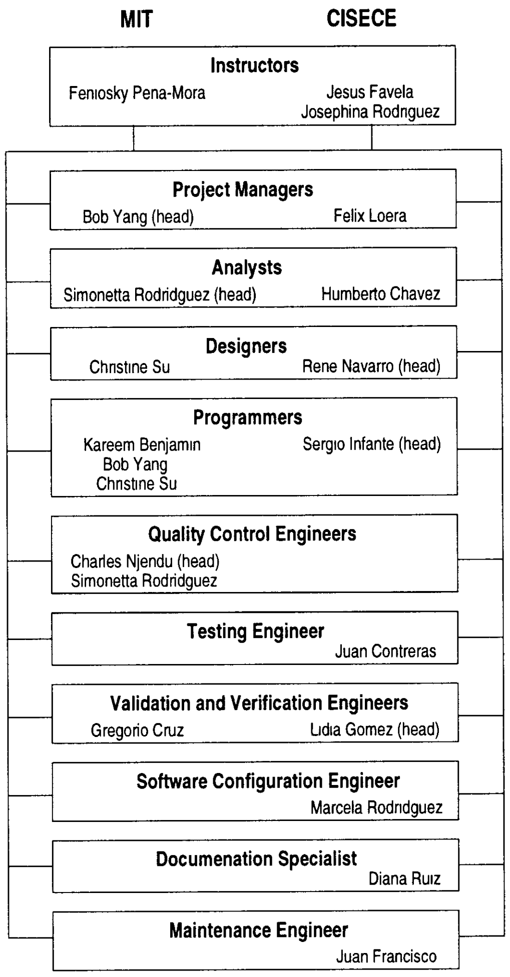


Figure 3-1: Participant Organization Chart

Objective of Design

The purpose of design is to create a clean and relatively simple internal structure, sometimes also called an architecture. A design is the end product of the design process. Thus, one goal of software design is to derive an architectural rendering of a system. This rendering serves as a framework from which more detailed design activities are conducted.

Good software architectures tend to have several attributes in common:

- They are constructed in well-defined layers of abstraction, each layer representing a coherent abstraction, provided through a well-defined and controlled interface, and built upon equally well-defined and controlled facilities at lower levels of abstraction.

- There is a clear separation of concerns between the interface and implementation of each layer, making it possible to change the implementation of a layer without violating the assumptions made by its clients.
- The architecture is simple, common behavior is achieved through common abstractions and common mechanisms.

The flow of information during the design phase is illustrated in Figure 3-2. Software requirements, manifested by informational, functional, and behavioral models, feed the design step. Using one of a number of design methods, the design step produces a data design, an architectural design, and a procedural design. The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software. The architectural design defines the relationship among major structural components into a procedural description of the software. Then source code is generated, and testing is conducted to integrate and validate the software.

The importance of software design can be clarified with a single word - *quality*. Design is the place where quality is fostered in software that can be assessed for quality. Design is the only way that one can accurately translate a customer's requirements into a finished software product or system. Software design serves as the foundation for all software engineering and software maintenance steps that follow. Without design, one risks building an unstable system that will fail when small changes are made. It also causes numerous difficulties in testing and quality assessment later in the software engineering process.

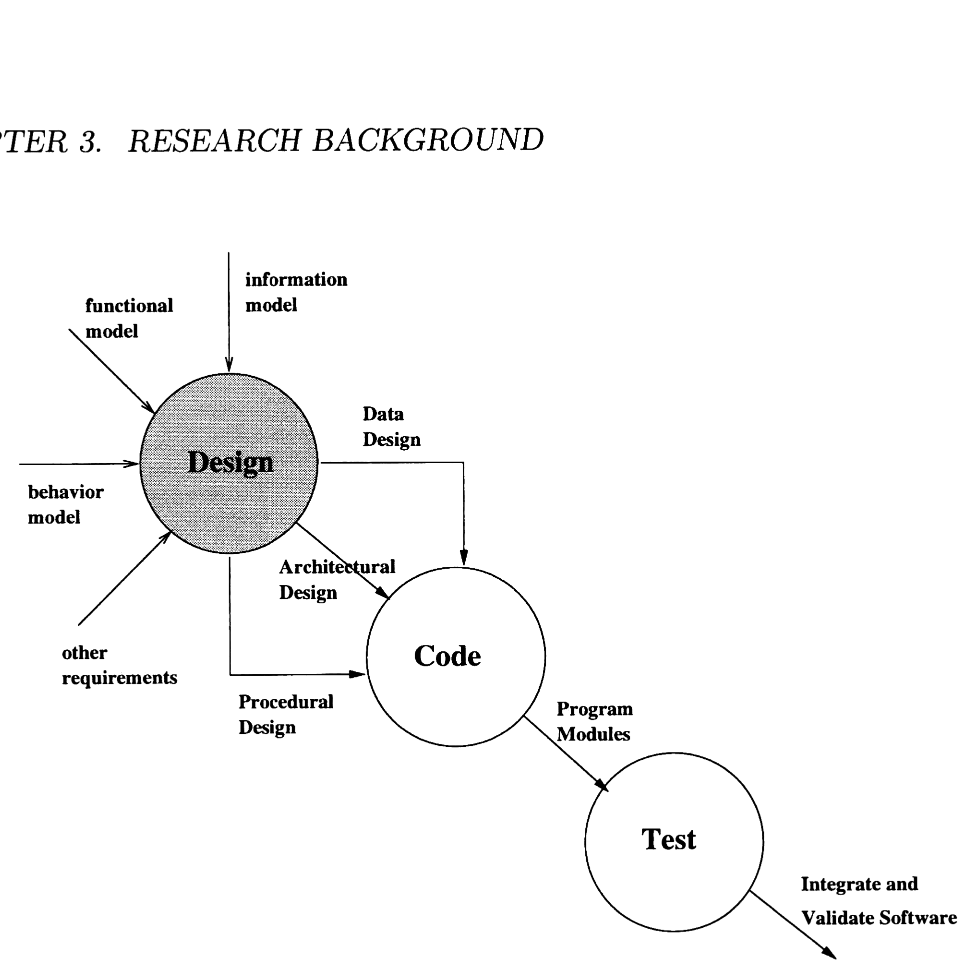


Figure 3-2: Design, Coding and Testing Flow Diagram, Adapted from [24].

Activities and Goals of Design

In order to evaluate the quality of a design representation, we must establish technical criteria for good design. The following guidelines may be useful, also see the items summarized in Table 3.1:

1. A design should exhibit a hierarchical organization that makes intelligent use of control among elements of software.

2. A design should be modular, that is, the software should be logically partitioned into elements that perform specific functions and subfunctions [4].

3. A design should contain both data and procedural abstractions.

4. A design should lead to modules (e.g., subroutines or procedures) that exhibit independent functional characteristics.

Table 3.1: Activies and Goals of Design Phase

Activity	Goal
Subsytem Decomposition	Create a system internal structure, so called an architecture and definition of relations among subsystems.
Set global resources access management	To select the appropriate policies for logical namimg, space, physical units, and shared data access.
Choose data storage management technique	To select the appropriate storage method for data structures. e.g. Data structures vs file system vs DBMS
Interact with the programmers	To select the appropriate language and programming paradigm
Subsystem allocation to processors	Assign processes to processor units that serves as a platform for subsystem execution
Concurrency management	Identify those case where system execution involves multiple threads of control
Control strategy selection	Determine appropriate method for lines of execution control. e.g. Procedural vs Event driven vs Concurrent
Boundary conditions management	Ensure that modules operate properly at boundaries established to limit or restrict processing. Initializations, termination, and failures

5. A design should lead to interfaces that reduce the complexity of connections between modules and with external environment,

6. A design should be derived using a repeatable method that is driven by information obtained during software requirements.

Design Decomposition

Software design is conducted in two steps [24, pages 318-321]. Preliminary design is concerned with the transformation of requirements into data and software architecture. Detail design focuses on refinements to the architectural representation that lead to detailed data structure and algorithmic representations of software. In addition to data, architectural, and procedural design, many modern applications have a distinct interface design activity. Interface design establishes the layout and interaction mechanisms for human-machine interaction.

Object-Oriented Design

Object-Oriented Design (OOD) encompasses the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design; specifically, this notation includes class diagrams, object diagrams, module diagrams and process diagram [9].

Here are some information about principal OOD methodologies:

- **Booch:** This method defines different models to describe one's system. The logical model or problem domain is represented in the class and object structure. In the class diagram, one can construct the architecture. The object diagram shows how the classes interact with each other, it captures some moments in the life of the system which helps to describe the dynamic behavior.[6]

- Fusion: A systematic software development method for object-oriented software that was developed at Hewlett-Packard Laboratories in Bristol, England. The method integrates and extends the best features of earlier methods, including OMT, Booch and CRC. Fusion is a full-coverage method, providing a direct route from a requirements definition through analysis and design to a programming language implementation.[12]

- Shlaer-Mellor: A well-defined and disciplined approach to the development of industrial-grade software. It is based on the object-oriented paradigm, and has developed over the past dozen years in the pragmatic environment of real-world projects. These projects have included manufacturing and process control applications, intelligent instruments and peripherals, banking operations, telecommunications, and defense applications.

- OMT: An object-oriented design technique that provides a method for representing software design. The method consists of several notations, one of them being a diagramming technique for representing classes and their relationships. As such, it has built-in concepts for attributes, class inheritance, and class relationships.

- Unified Modeling Language: An application modeling language for use-case modeling, class and object modeling, component modeling, and distribution and deployment modeling.

Relation with the other Roles

Through the software development process we observe a high degree of interaction among system design and the different key process areas.

Analyst Designer translates specification of requirements established in requirement analysis into an model of implementation. Interact with the analyst to determine project feasibility. Usually analyst assists designer and vice versa.

Programmer Designer creates the system implementation blueprints for programmers. This model is translated into a machine readable form during encoding process. Designer assists programmer in programming language selection and interpretation of design documents such diagrams, charts, tables, etc. etc.

Test Engineer Designer coordinate efforts with test engineer in order of assure that architectural design of software system includes specifications that helps in test cases exercise. Assists test engineer in requirements verification.

Quality Control Quality control engineer reviews design phase in order of ensure design process products quality and fulfills performance, design, and verification requirements.

Validation and Verification Engineer Validation and verification engineer assessments the level of accordance between client requirements and the system's model of implementation designed, looking for misunderstood, missing or wrong implemented features.

Documentation Specialist Documents specialist keep design documents once that design process is completed and makes this document available for the rest members of the development team.

Software Configuration Manager During design, software configuration manager controls design changes and maintains complete records of every change and its rationale.

Maintenance Engineer Designer assists maintenance in managing of post delivery evolution. This evolution comprises bug fixes, system functionality enhancements, and requirements modifications.

Project Manager Designer works under coordination of project manager in order of build a system architecture that meets requirements under given budget constrains and availability of human resources. Additionally, project manager uses design specifications for planning and estimate resource allocation.

Tools

There are many valuable software design tools available on the market, they generally provide vast functionalities of planning and process complex software projects. Table 3.2 lists some popular CASE toolsets that are used in a wide spectrum of platforms.

Designer Profile: Who should play the role?

- For small and medium size systems, architectural design is typically the responsibility of one or two particularly insightful individuals. They must have that unusual ability to synthesize workable solutions amidst a myriad of constraints.

Table 3.2: Tool Categories

Company	**Tool Name**	**Platforms Supported**	**Description and Methodologies Supported**
Hewlett Packard	HP, C++/Softbench	HP	CASE tool integration, C/C++ development
Iconix Software Engineering	Iconix Power Tools	Macintosh, Windows, Dos, HP, Sun, SGI	Multiuser, OO development toolset, OMT and Booch
Mark V Software	ObjectMaker	MS-Windows, Unix, Macintosh	Object-oriented analysis and design, Booch
Popkin Software	System Architect	MS-Windows, OS/2	Object-oriented design design, Booch, Shlaer/Mellor
Platinum Technology	Paradigm Plus	Windows, Unix, OS/2	CASE toolset supporting OMT, FUSION, Booch, Shlaer/Mellor and Customized methods
Rational	Rose	Unix, AIX	Object-oriented analysis and design Booch, OMT, and UML

- Designers are usually the best qualified to make strategic decisions due their previous experience in building similar systems.
- Designers are no necessarily the most senior developer.
- Designers should have adequate programming skills.
- Designers should be well versed in the notation and process of object-oriented development.

Work Plan

A successful project can not come without a work plan. The work plan is a guide line for development, and it is a yard stick for progress measurement. The design phase of the DISEL project lasted about 40 days. It started on January 10, 1998, and ended

Table 3.3: Work Plan and Time Table

Activities	**Time (Days)**
System Design	20
Organize the system into subsystems	3
Identify concurrency inherent in the problem	2
Allocate subsystem to processor and task	2
Choose data storage management	2
Identify global resources and determine access mechanisms	3
Choose an approach for implement execution control	5
Consider boundary conditions	3
Object Design	20
Obtain operations for the object model from other models	7
Design algorithms to implement operations	1
Optimize access paths to data	1
Implement software control	3
Verify class structures to increase inheritance	2
Design implementation of associations	3
Determine the exact representation of object attributes	2
Package classes and associations into modules	1

on March 22, 1998. Table 3.3 is a detailed listing of the activities conducted within the design phase.

3.2.5 Object Oriented Approach

Like any specialized field of research, object technology has accumulated its own set of terms and definitions. This specialized language is often an obstacle for new people attempting to understand objects. Thus, the following section is provided to pave away such obstacles and to help readers to become familiar with object technology terms.

Object Oriented Concepts

A software object is used to represent some real world entity, such as a part number or an address, which the system must manipulate. An *object* is defined as a software package which contains a set of related data, and all the functions and procedures needed to access and modify that data. The data is often referred to as the object's state. The functions are called the object's *methods.* Calling one of an object's methods is often referred to as sending a *message* to that object, requesting that the object provide some services.

Classes

Programs rarely require a single object with unique state and methods. It is much more common to manage collections of objects which share methods and state, and differ only in the values of their state variables. If two or more objects share the same methods and state variables, they are said to be members of the same *class.* They might also be referred to as *instances* of that class. For example, a bank might have an account object which keeps track of the account number, owner's address and balance. Each real account held by the bank would be represented by a single instance of the account class.

It is important to distinguish between a class and an object. A class is simply the definition of what data is stored in instances of that class and what operations are available for manipulating that data. An object is the item which stores the data, and exists inside the program. A class can be thought of as the blueprint and operating instructions for its objects.

Inheritance

An application will often manipulate several very similar classes of objects. For example, the previous banking application (see Section 3.2.5) might have checking, savings, and money market accounts. If each of these were implemented as a class, there would be much overlap between the three classes. They would share state variables like account number, owners name, address, and balance. They would also share many methods like deposit, withdrawal, and add co-signer.

With object oriented languages, the programmer can eliminate this redundancy by creating a class which contains all the shared state and methods. The programmer can then use this base class to create more specialized classes. This is called *inheritance.* The generalized top level class is called the *superclass.* The more specialized classes are called *subclasses.* More detailed discussions of inheritance are provided later in Section 3.4.1, please refer to it for further clarification.

Polymorphism

In procedural languages, each identifier must have a unique meaning. A variable name can only refer to a single datum. No two functions can share the same name. This is not true in object oriented languages. In an object system, two or more classes can use the same name for different methods. When one of these methods is invoked, the object system uses the class of the object on which the method is being invoked to determine which method to use. This reuse of names is called *polymorphism.* Because it reduces the number of similar and sometimes redundant function names, and hides the class of the object from the calling routine, polymorphism makes programs easier to write and easier to understand. Polymorphism can also be achieved through hierarchical forming and the binding is created dynamically at run time.

3.3 Object Oriented Design

Object oriented Design (OOD) creates a representation of the real-world problem domain and maps it into the software solution domain. OOD results in a design that interconnects data objects and processing operations in a way that modularizes both information and processing [8].

It is known that there exist many different programming styles, each inevitably relates to a specific language. A programming style is a way of organizing programs on basis of some conceptual model of programming, in return, language that is used will make such style clear. There are five main kinds of programming styles, and each comes with its own kind of abstraction (See Table 3.4). Each of these styles

Table 3.4: Programming Styles and Abstractions

Programming Style	**Applicable Abstraction**
Procedure-oriented	Algorithm
Object-oriented	Classes and objects
Logic-oriented	Goals
Rule-oriented	If-then rules
Constraint-oriented	Invariant relationships

of programming is based upon its own conceptual framework. Each requires a different set of rules in approaching the problem. For object-oriented, the framework is the object model. The object model will possess four characteristics: abstraction, encapsulation, modularity, and hierarchy. Each property is required for good object modeling.

3.3.1 Abstraction

Abstraction is one of the fundamental ways to cope with complexity. "Abstraction arises from a recognition of similarities between certain objects, situations, or processes in the real world, and the decision to concentrate upon these similarities and to ignore for the time being the differences" [10, page 83]. A good abstraction is one that emphasizes details that are significant to the reader or user and suppresses those that are irrelevant or unimportant. An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer [5].

An abstraction serves as an outside view of an object, so it aims at separating an object's essential behavior from its implementation. Such a separation is called an abstraction barrier, achieved by applying the principle of least commitment, through which the interface of an object provides its essential behavior [1]. Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer. Figure 3-3 is an illustration of this definition.

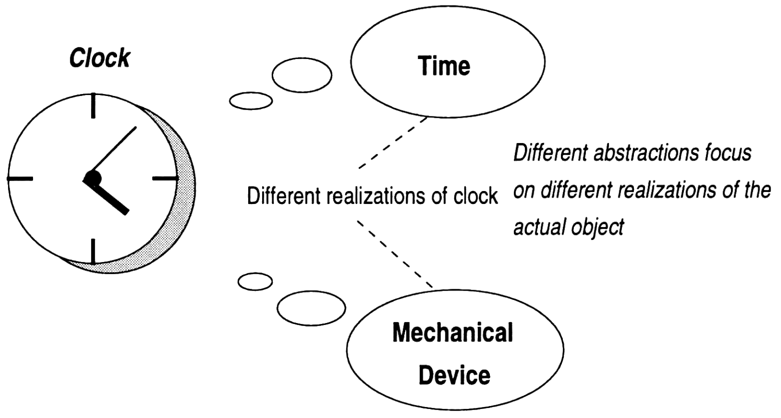

Clock
Time
Different realizations of clock
Different abstractions focus on different realizations of the actual object
Mechanical Device

Figure 3-3: Abstraction Representation

3.3.2 Encapsulation

The concept of encapsulation and abstraction comes hand in hand. Abstraction focuses upon the outside view of an object and encapsulation prevents others to obtain the inside view, where the behavior of the abstraction is implemented. As Ingalls suggests, "No part of a complex system should depend on the internal details of any other part"[16, page 9]. In this manner, encapsulation provides explicit barriers among different abstractions or independent modules.

In practice, a class should have two parts, an interface and an implementation. The interface of a class captures only its outside view, encompassing abstraction of the behavior common to all instances of the class. The implementation of a class comprises the representation of the abstraction as well as the mechanisms that achieve the desired behavior. The explicit division of interface/implementation represents a clear separation of concerns. To sum up, encapsulation is the process of hiding all the details of an object that do not contribute to its essential characteristics,it has important ramifications in software maintainability and reuse. Figure 3-4 depicts the concept of object encapsulation.

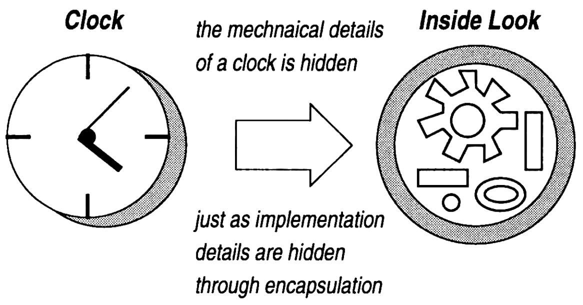


Figure 3-4: Encapsulation Representation

3.3.3 Modularity

Modularity is the property of system that has been decomposed into a set of cohesive and loosely coupled modules. In traditional structured design, modularization is primarily concerned with the meaningful grouping of subprograms, using the criteria of coupling and cohesion. In object-oriented design, the problem is subtly different: the task is to decide where to physically package the classes and objects [25].

The goal of decomposition into modules is the reduction of software cost by allowing the modules to be designed and revised independently. Each module should be an independent entity, and the change of implementation will not affect the behavior of the others. In connection with the other two properties, an object provides a clear boundary around a single abstraction, and both encapsulation and modularity provide barriers around this abstraction. See the following illustration (Figure 3-5) for an example of modularity.

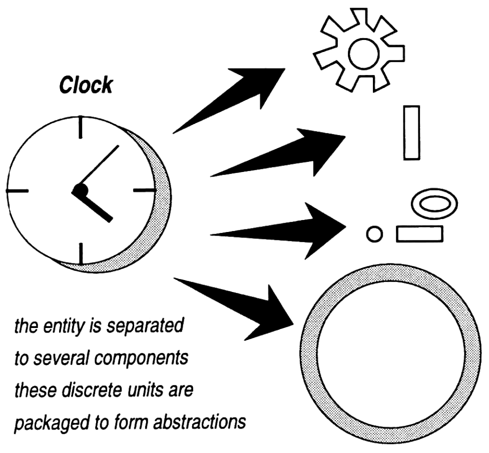


Figure 3-5: Modularity Representation

3.3.4 Hierarchy

When dealing with complex system, it is often necessary to understand it in an organized fashion and this view is hierarchical in some sense. A set of abstractions will form a hierarchy, and by identifying these hierarchies in the design, one can greatly simplify the understanding of the problem [29].

There are two important hierarchies exist in a complex system: class structure and object structure. Among class structure, inheritance is the one essential concept. Inheritance defines a relationship between classes, where one class shares the structure or behavior defined in another class. Inheritance represents a hierarchy of abstractions, in which a subclass inherits attributes and methods from one or more baseclasses. Typically, a subclass augments or redefines the existing structure and behavior of its superclass.

The other type of hierarchy can be represented by aggregation. For example, a car object is built up of four wheels, an engine, and frame. In terms of inheritance, a high-level abstraction is generalized, and a low-level abstraction is specialized. In terms of aggregation, a class is at a higher level of abstraction. The following figure (Figure 3-6) depicts the hierarchies described here.

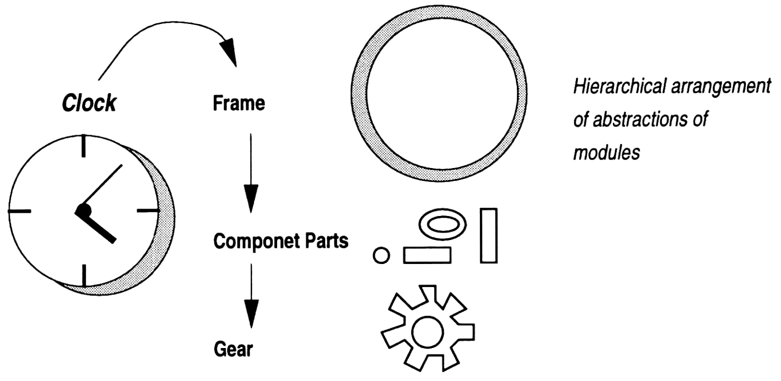


Figure 3-6: Hierarchy Representation

3.4 Unified Modeling Language

The Unified Modeling Language (UML) is a third generation method for specifying, visualizing, and documenting the artifacts of an object-oriented system under development. The UML represents the unification of the Booch, Object Modeling Technique (OMT) methods and Jacobson. The UML is the direct and upward-compatible successor for both Booch, OMT and Jacobson. By unifying these two leading object-oriented methods, the UML provides the basis for a de facto standard in the domain of object-oriented analysis and design. The following sub-sections will provide a walk through of the UML notations. These set of notations are separate in six categories, each comes in a form of diagram representation. They are the class diagram, the use case model, the message trace diagram, the state diagram, the module diagram, and the platform diagram.

3.4.1 Class Diagram

The class diagram is the core of a Unified Model. This view shows the logical static structure of a system: its contents and their relationships to each other. In its most fundamental form, it shows the elements composing the state of a system. Class diagrams show generic descriptions of possible systems and object diagrams show particular instantiation of systems and their behavior. Class diagram contain classes and object diagrams contain objects, but it is possible to mix classes and objects for various purposes, so the separation is not rigid.

Class

A class is a descriptor for a set of objects with similar structure, behavior, and relationships. It represents a concept in the system being modeled. Class diagrams are static structures that show entities that exist, their internal structure, and their relationship to other identities. The notation for a class is a solid-outlined rectangle with three compartments. The top compartment holds the name of the class and any other of its general properties such as stereotypes. The middle compartment holds a list of the class's attributes and the bottom compartment holds a list of the class's operations. The visibility of an attribute or operation is shown in public (+), protected (#), and private (-) format below in Figure 3-7.

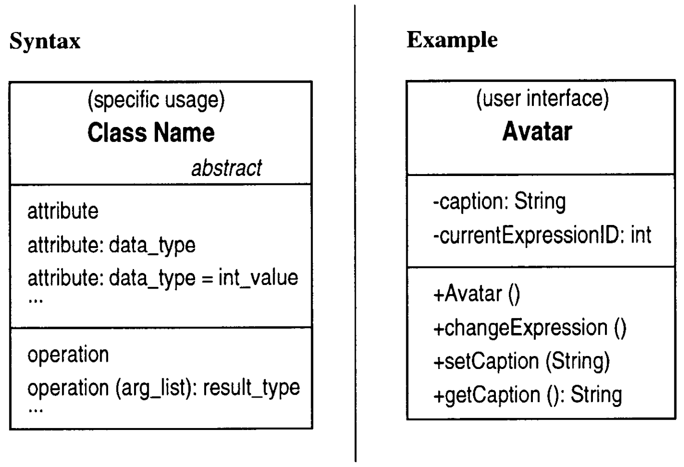


Figure 3-7: Class Diagram

The UML specification suggests a number of formatting styles for class diagrams. It suggests that the name of the class should be centered within the compartment and in boldface, where as the names of abstract class should be shown in italics.

The name and general properties of the class can be displayed in up to three sections inside the class name compartment. The topmost section holds an optional stereotype keyword or icon. Below that is the name of the class, and the lowest section contains a list of properties in braces. These properties are class-level attributes that cannot be expressed using normal UML syntax, (e.g. {author="Christine Su"})

The other compartments within the class diagram hold strings that represent a feature such as an attribute or operation of the class. These features are kept in a list and the order of the list is important. An attribute of a class is a type expression that describes a property of the class. Attributes are also known as data members or member variables. The recommended syntax of an attribute is:

public name: type-expression = initial-value {property=string}

An operation of a class is a method, also called a member function, that is supplied by that class. The recommended syntax for an operation is:

public name (parameter-list): return-type-expression {property-string}

Inheritance

Inheritance is the taxonomic relationship between a superclass and its subclass. It is often called generalization or specialization. It is shown in UML by a solid path between two classes. A hollow triangle icon placed at the end of the path attached to the class whose characteristics are being inherited (See Figure 3-8).

Object

An object is an instance of a class. The UML notation for an object is a hexagon with two compartments. The top compartment contains the name of the object and its class in the format of *objectName: className*. Objects have the same stereotype as their class. The second compartment holds the attributes of an object in the following format: *attributeName : type = value*. An object can have a multiplicity

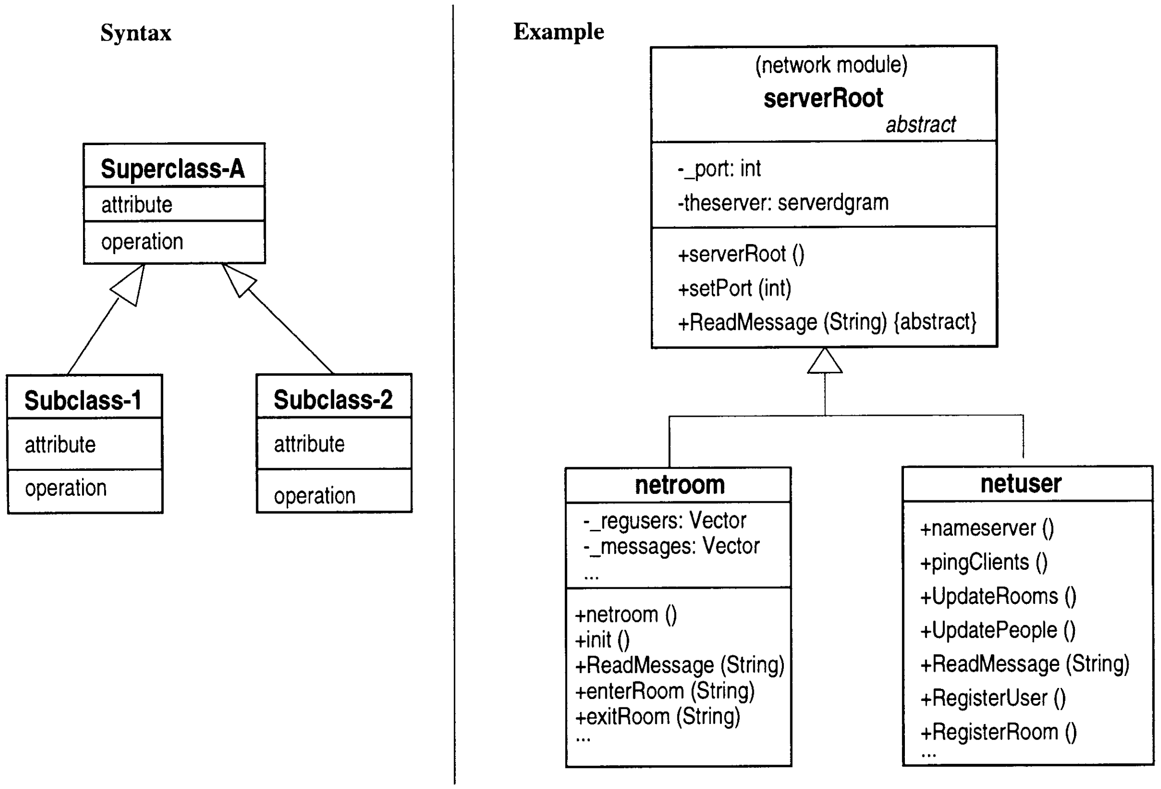


Figure 3-8: Inheritance Illustration

within an enclosing composite class. The multiplicity indicator is drawn as a small integer-range expression in the peak, such as 3, 5, 7..13, and 19..*. This indicates how many instances of the class can exist at a time. The symbol "*" indicates no upper bound is set (See Figure 3-9).

Template

UML includes syntax for *parameterized* or template classes. A template class has one or more unbound formal parameters. These parameters create a family of classes which are specified by the binding of parameters to values. It is also possible to specify operations as parameters of template classes. Class templates are not directly usable, they must be instantiated before objects can be made from them. In addition, class

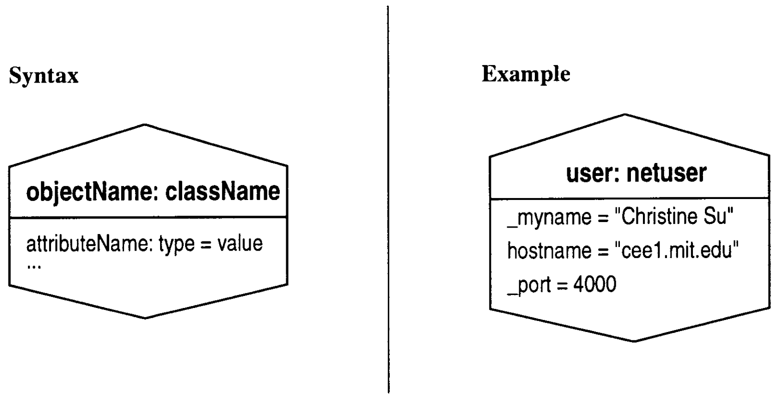


Figure 3-9: Object Notation

templates cannot be superclasses or the target of an association. They may, however, be a subclass of a regular class. The notation for a class template is a regular class rectangle with a dashed rectangle superimposed on the upper right corner of the class rectangle. This dashed rectangle contains the parameter list (See Figure 3-10). The result of an instantiated class template is called a bound element. It has the following notation:

template-name <value-list>

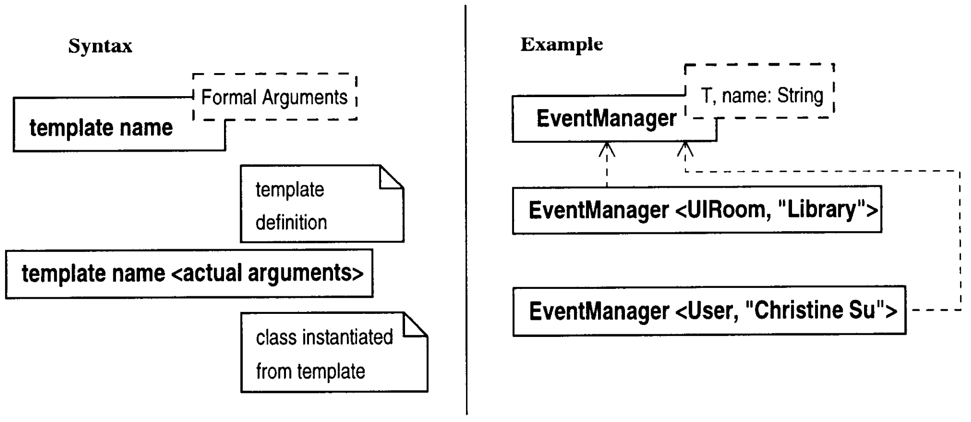

Syntax
template name
Formal Arguments
template definition
template name <actual arguments>
class instantiated from template
Example
EventManager
T, name: String
EventManager <UIRoom, "Library">
EventManager <User, "Christine Su">

Figure 3-10: Template Notation

Package

A *package* is a grouping of model elements such as classes. Packages own the model elements they contain and may be nested within one another. A package is shown as a large rectangle with a tab attached to one of its upper corner. If the contents of the package are shown in the symbol, the name of the package is placed in the tab. Otherwise, the name is placed in the main compartment of the package symbol. A package can be imported by using the *imports* dependency. A dependency dashed-line arrow is drawn from the referencing client package to the target supplier package. The following diagram will depict such a relationship between packages (See Figure 3-11).

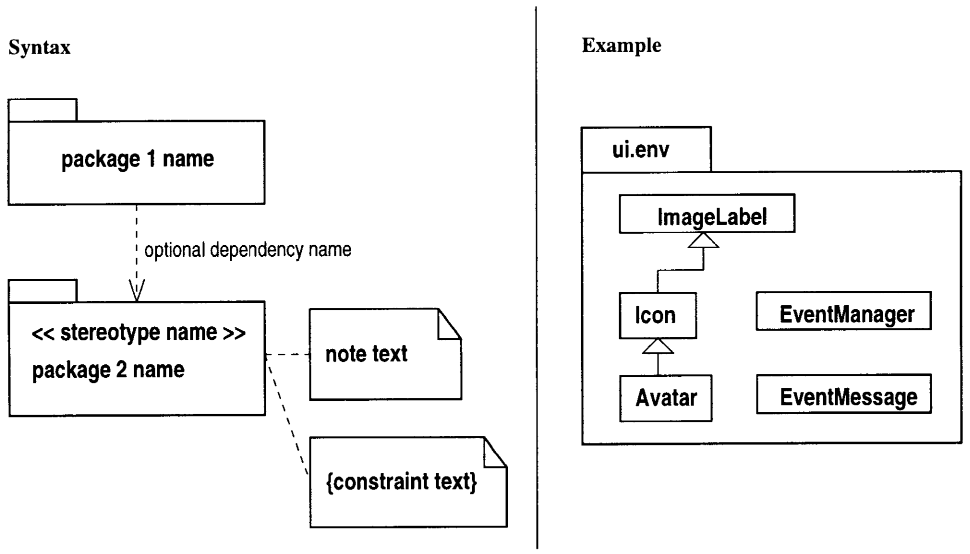


Figure 3-11: Package Notation

Interface

UML also supports a mechanism for representing *interfaces* among classes. An interface specifies externally-visible operations of a class or component without exposing its internal structures. Interfaces contain no implementation, only operations. An interface is noted by a rectangle symbol with compartments and the keyword *interface.* A dashed line with an arrowhead pointing to the class that supports it is used to indicate that a class realizes an interface. In addition, an interface involve in an association may be shown with a small circle with the name of the interface placed below it. The circle may be attached via a solid line to classes or packages that support it. The class that uses the operation of the interface are attached to the circle by a dashed arrow pointing to the circle. Figure 3-12 below illustrates the concept of an interface.

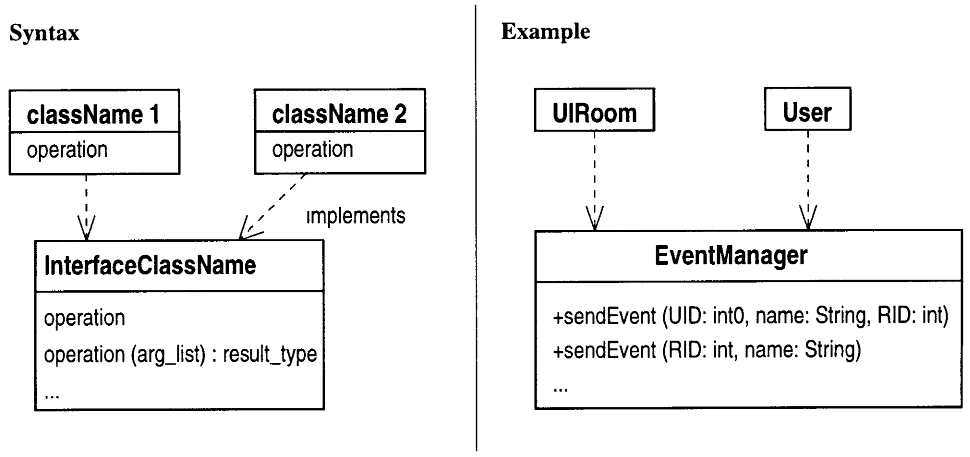


Figure 3-12: Interface Notation

Association

Association represent structural relationships between objects of different classes, information that must be preserved for some duration and not simply procedural dependency relationships. The individual instances of an association are called *links.* Most associations are binary, drawn as solid lines between pairs of classes.

An association could have different names in each direction. The name may be placed on or adjacent to the association line. The name of an association can be omitted, particularly if rolenames are used. Each end of an association is a *role.* Each role may have a *rolename*, showing how its class is viewed by the other class. The rolenames must be unique, and it is placed next to the end of the line. Multiplicity is indicated by a text expression on the role. A range is indicated by the lower bound, an integer, followed by two dots and another integer for its upper bound, e.g. 0..*.

An *association class* is an association whose elements have attribute values or operations. It is shown by drawing a dashed line from the association line to a class box that holds the attribute, operations, and associations if they exist. The class box can contain multiple attributes and operations to apply to the original association (See Figure 3-13).

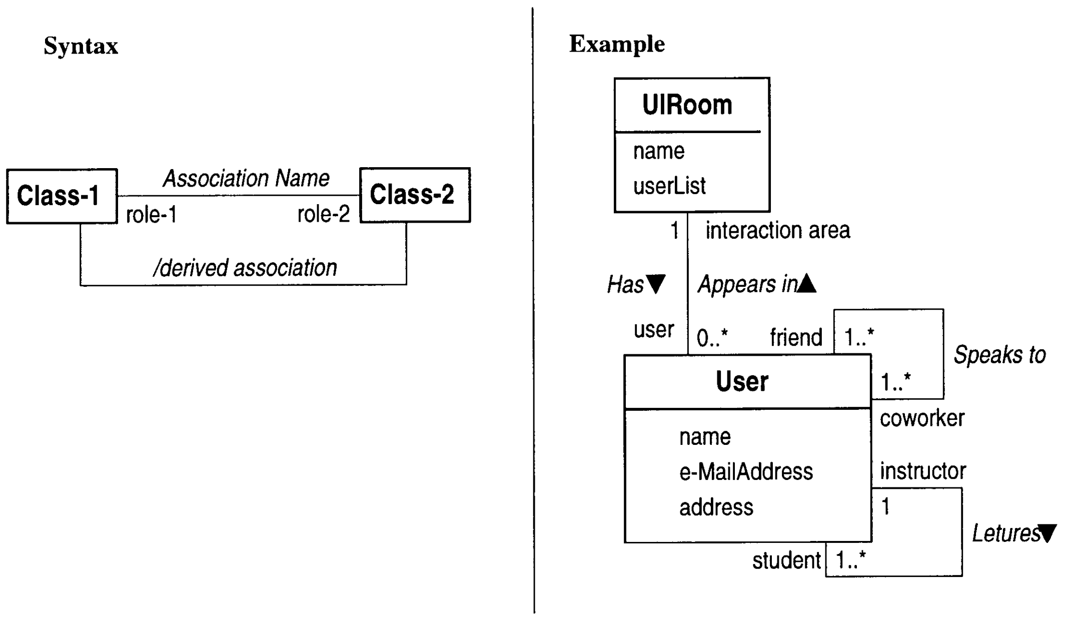


Figure 3-13: Association Class Notation

A *qualified association* is a variant form of association attribute. A *qualifier* is an association attribute value that is unique within the set of links associated with an object in the association. In other word, an object and a qualifier value identify a unique object across the association. Together, they form a *composite key*. The qualifier is part of one role of the association, and it does not need to be symmetric.

Qualifiers are drawn as small boxes on the end of the association attached to a class. They are part of the association, but not the class (See Figure 3-14).

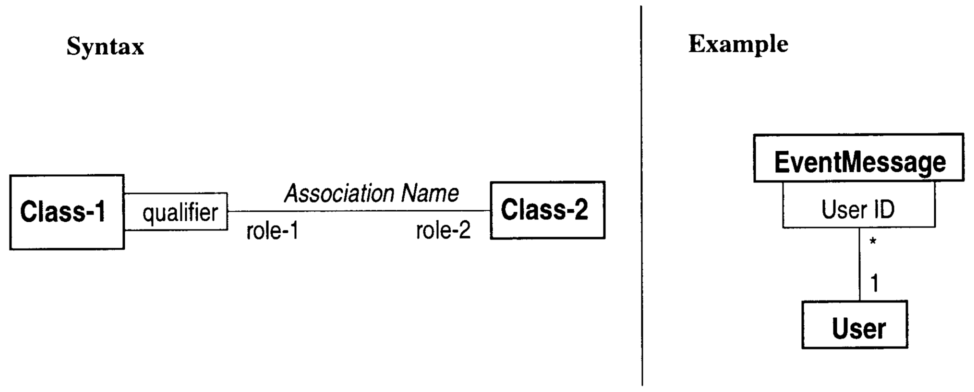


Figure 3-14: Qualified Association Notation

Aggregation is a special form of association. It indicates the lifetimes of the parts are dependent on the lifetime of the whole. It does not indicate a particular kind of implementation or navigation direction. It is drawn by placing a diamond on the role attached to the whole object. The multiplicity of an aggregate can be one or many. A multiplicity of one indicates a *physical aggregation*, where a multiplicity of many indicates a *catalog aggregation* (See Figure 3-15).

The *navigability* property on a target role indicates the implementation of the association from the originate role. It is normally implemented using a pointer from one class to another. It is a design and implementation property indicates by a small square placed on the target end of the association next to the target class. If the square is hollow, it is a by-reference implementation, if the square is solid, it is a by-value implementation (See Figure 3-15).

Ternary associations are drawn as diamonds with one line path to each participating classes. This is the traditional entity-relationship model symbol for an association. Since most uses of ternary associations can be eliminated by using qualified associations, its usages are rare (See Figure 3-16).

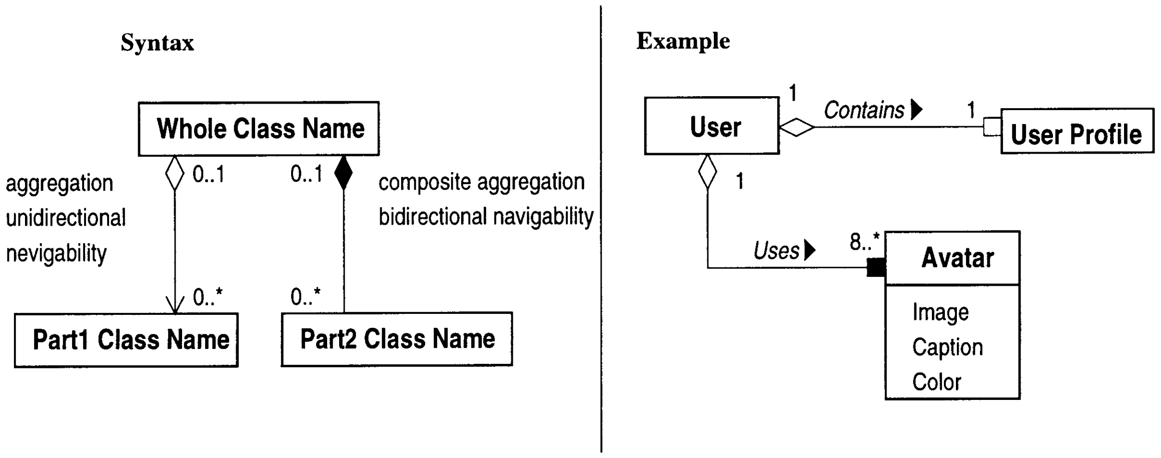


Figure 3-15: Aggregation and Navigability

Constraint

A *constraint* is a restriction on values expressed as an arbitrary indicator attached to a class or association. A constraint may be written as text within braces, either free standing or embedded in a note attached to the affected elements. For binary constraints, a dashed dependency line may be drawn between the affected elements, the line is labeled with the constraint string enclosed in braces. Navigation expressions are handy for writing constraints, and notes can be used for placing arbitrary comments on diagrams or for showing implementation code for operations (See Figure 3-17).

Composite

A *composite* is kind of pattern or macro that represents a conceptual clustering for a given purpose. Composition is shown by drawing a class box around its embedded components, which are prototypical objects and links. A composite defines a context in which references to classes and associations defined elsewhere can be used. A composite is a class that has identity. All the objects and links constrained within a composite box take identity from the composite, and they belong to the same

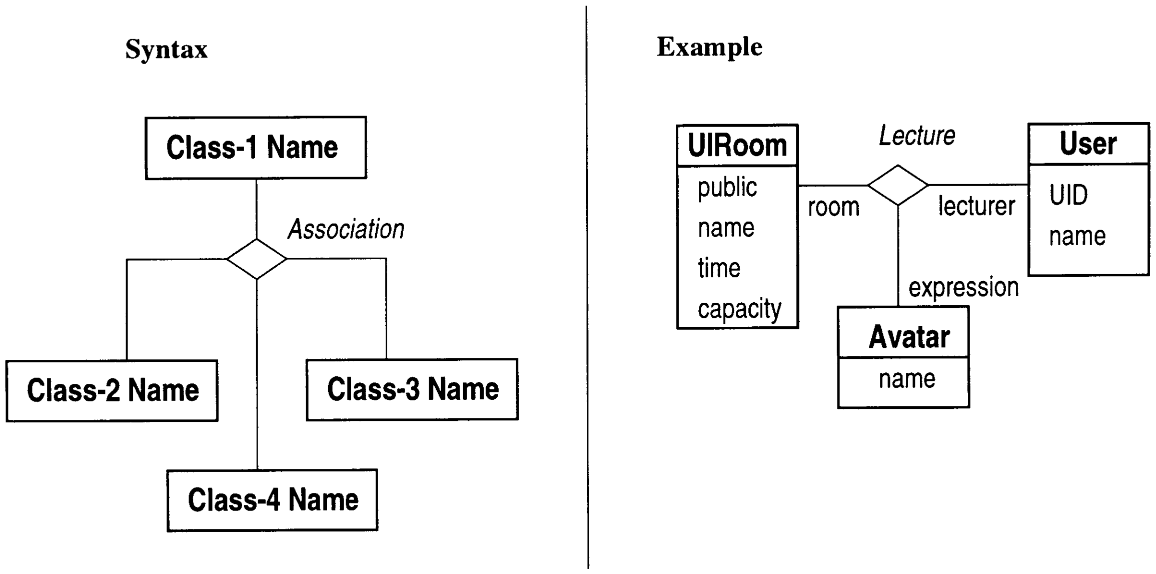


Figure 3-16: Ternary Association

instantiated object (See Figure 3-18).

Category

A *category* is a subset of the model itself. Each category owns some of the classes, associations, and generalization of the model. Categories are purely organizational, they have no logical semantics. A category is drawn as a rectangular box with a double

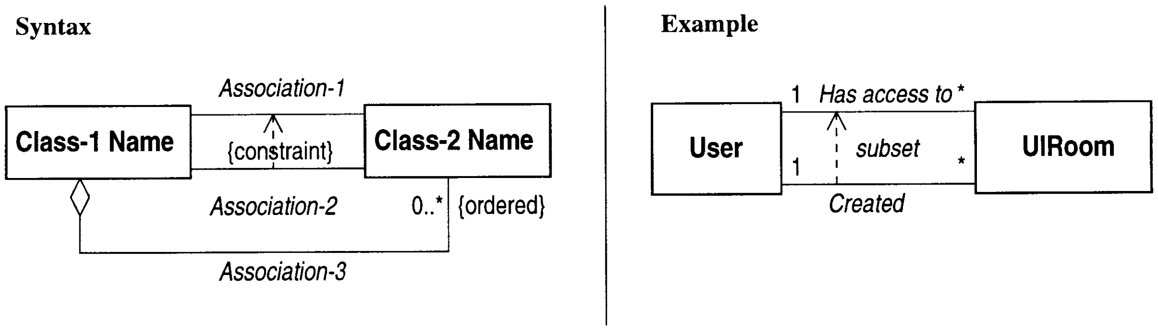


Figure 3-17: Constraints

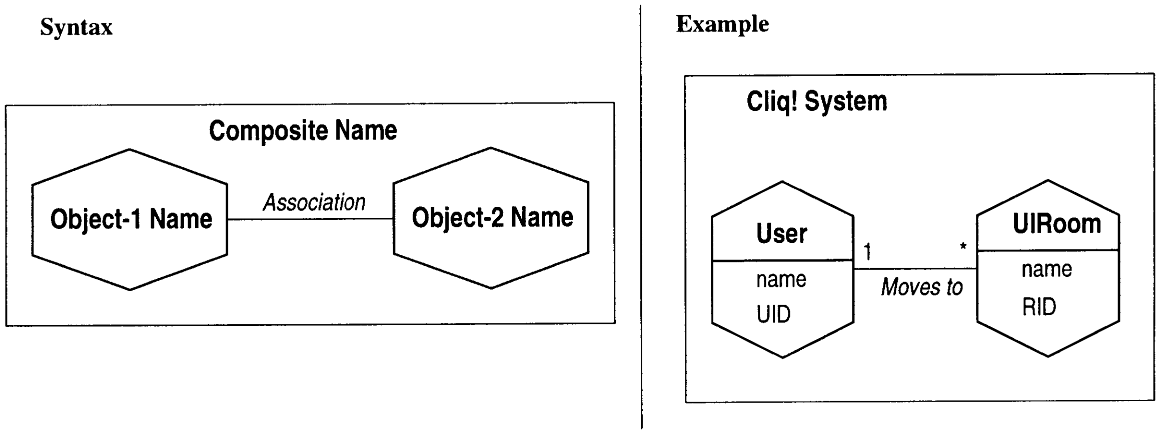


Figure 3-18: Composite Representation

outline. Dependencies between categories are shown by dashed arrows between them, with the tail on the client and the arrowhead on the supplier (See Figure 3-19). A *category diagram* is a kind of class diagram showing only categories. Decomposition of a category into smaller categories can be shown in two ways: by nesting categories inside large ones, or by using the aggregation syntax. A *category interface diagram* is a kind of class diagram showing categories and their public classes and relationships. As an organizational element, categories provide the home for other modeling technique, such as the use cases which will be discussed later in Section 3.4.2.

3.4.2 Use Case Model

A *use case* is a generic description of an entire transaction involving several objects. A use case can describe the behavior of a set of objects, such as an organization. A use case model thus presents a collection of use cases and is typically used to specify or characterize the behavior of a whole application system together with one or more external actors that interact with that system [19].

An individual use case may have a name, and its meaning is often written as informational description of the external actors and the sequences of events between

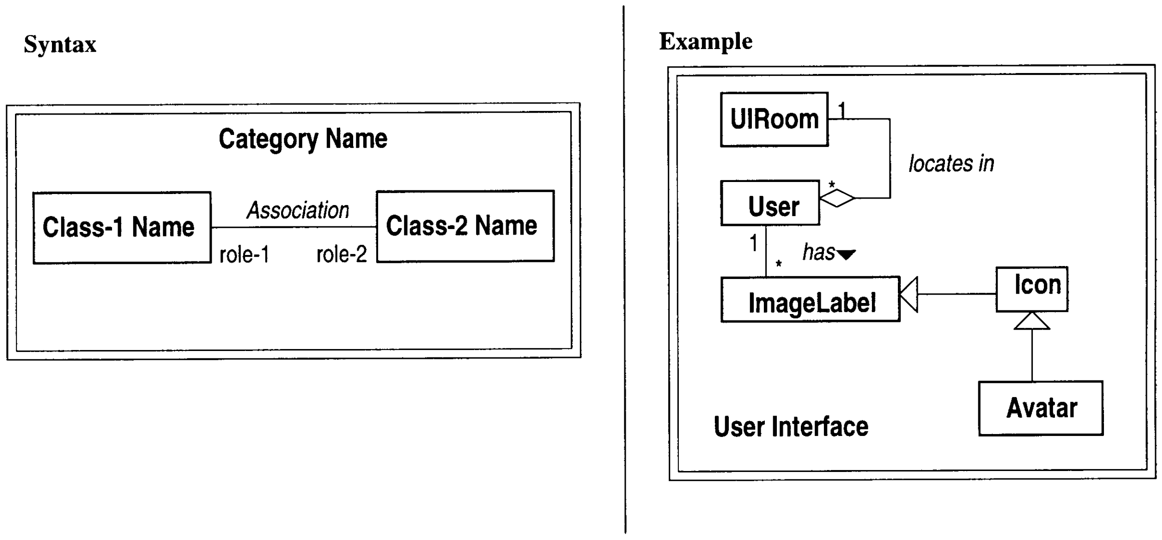


Figure 3-19: Category Representation

objects that make up the transaction. Instances of this behavior may be formally specified using scenarios, but interaction and conditionality within scenarios is usually best expressed as informal text.

A *scenario* is an instance of a use case. Each scenario provides a prototypical thread through its associated use case. Any given use case is typically characterized by multiple scenarios. Both use cases and their associated scenarios can be regarded as models for viewing purposes that can be derived from or built upon more fundamental models.

A top level *use case diagram* is helpful in visualizing the context of a system and boundaries of the system's behavior. A use case diagram shows the set of external actors and the system use cases that the actors participate in, as in Figure 3-20. A use case diagram contains a composite representing the system containing the use cases that it supports. Each case is connected to the external actors that use it. The use cases are drawn as ellipses within the icon for the system object. The actors are drawn as object icons outside the system icon. For further detail, each participation relationship can be labeled with message flows showing event exchanged between each

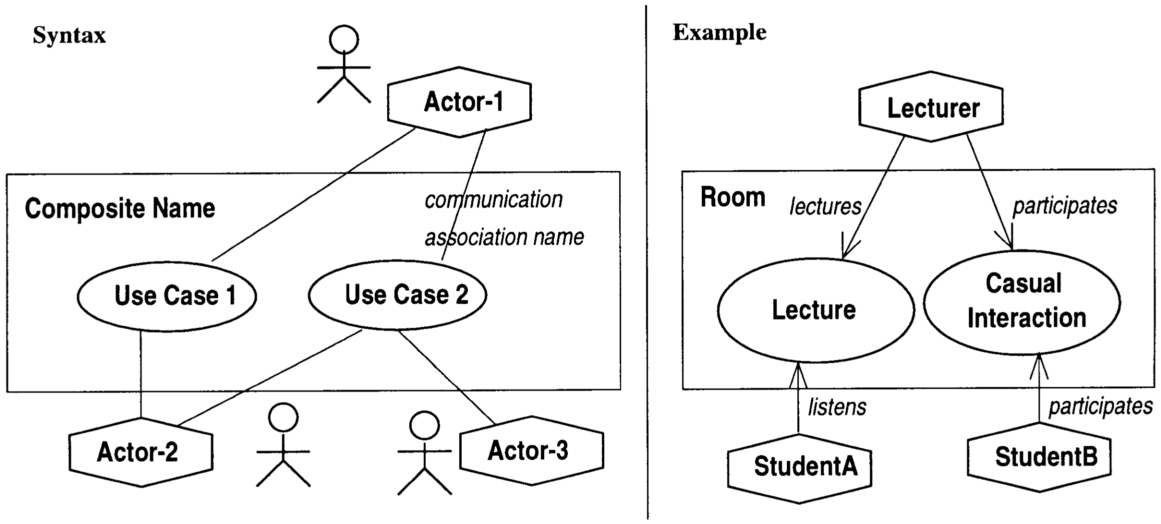


Figure 3-20: Use Case Representation

actor and each use case.

3.4.3 Message Trace Diagram

A scenario shows a particular series of interactions among objects in a single execution of a system. Scenarios illustrate interactions that are inherent in the underlying behavior of the associated objects but whose overall form is not apparent in their isolated behavior. State diagrams may be used for specifying the behavior of a class objects, whereas, scenarios are for understanding how such objects collaborate.

Scenarios can be shown in two different ways containing the same information but organized in different dimensions. A *message trace diagram* shows the interactions among a set of objects in temporal order, which is good for understand timing issues. A text dialog can accompany or replace such a diagram. An *object message diagram* shows the interactions among a set of object as nodes in a graph, which is good for understanding software structure, since the interactions that affect an object are localized around it. Ultimately, both forms build upon the same underlying semantics, and so it is possible to transform one view to the other without loss of information.

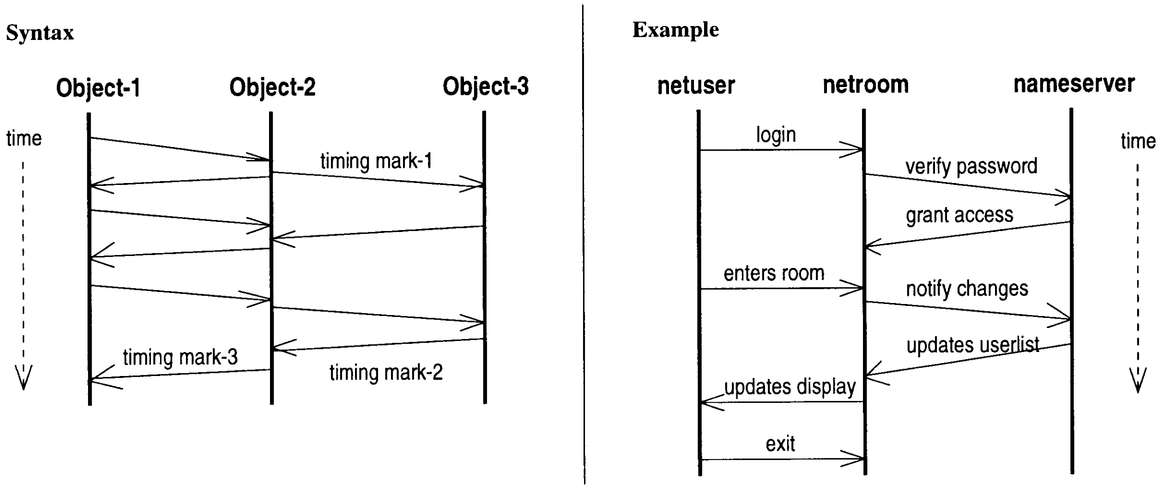


Figure 3-21: Message Trace Diagram Representation

In message trace diagram, objects in a transaction are drawn as solid vertical lines, their names are shown at the top. The line begins when the object is created and ends when the object is destroyed. An event or a message dispatch is drawn as a labeled horizontal arrow from the sending object's line to the receiving object's line. Time proceeds vertically, so event timing sequences can be easily seen. An object can send simultaneous events to other objects. To indicate events that require time to deliver, the event line can be tilted downward so that the sending and receiving times are distinct. *Timing marks* can be used to specify timing constraints and it is expressed as string labels that are placed along the event lines (See Figure 3-21).

3.4.4 State Diagram

The state diagram describes the evolution of an object of a given class in response to interactions with other objects inside or outside the system. Each class may have a state diagram to describe its dynamic behavior. Each state diagram is associated with one class or with a higher-level state diagram.

State diagrams are formal specifications of the behavior of a class. Scenarios are examples of execution of a system. They may involve several objects playing various roles. A state diagram is a directed graph of states connected by transitions. A state diagram describes all possible ways in which the objects respond to events from other objects. State diagram can also be used to show the life history of objects that undergo sequences of operations that take the objects into several fundamentally different states (See Figure 3-22).

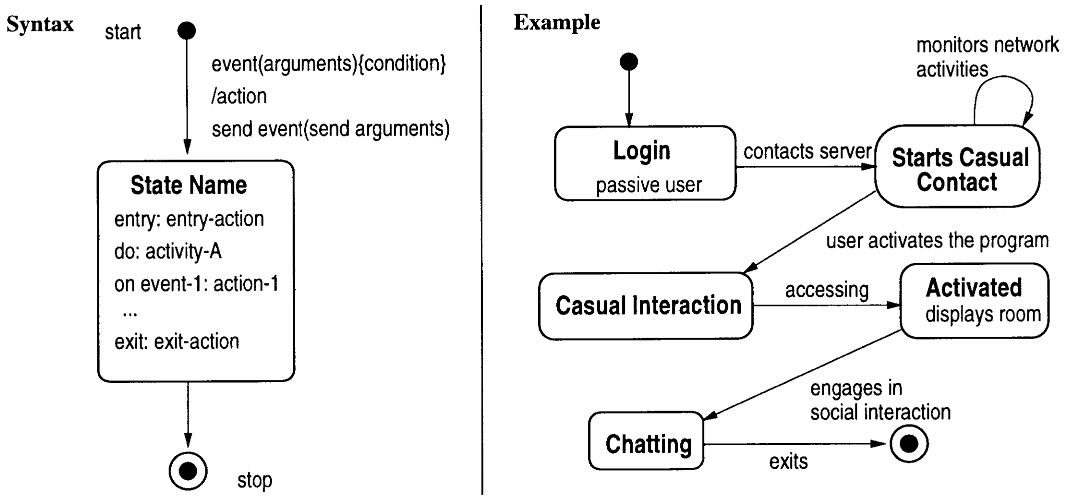


Figure 3-22: State Diagram Representation

3.4.5 Module Diagram

The development view of a system may be specified and visualized in module diagrams, which represent the physical modules that provide the defining occurrence of these logical elements. Modules may be distinguished as either specifications or implementations. Similar to the issues of scale addressed by categories in the logical model, the clusters of modules can be shown via subsystems. A *module diagram* is used to show the allocation of classes and objects to modules in the physical design of a system; a single module diagram represents all or part of the module architecture of the system [4, page 175] (See Figure 3-23).

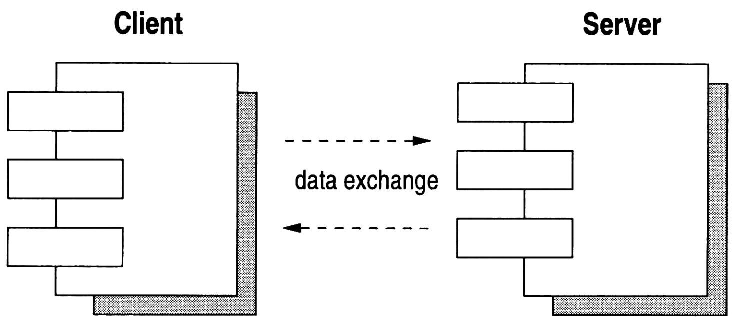


Figure 3-23: Module Diagram Representation

3.4.6 Platform Diagram

The physical topology upon which a software system executes may be specified and visualized in a platform diagram. Such a diagram includes processors and devices all united by connections along which information may pass. Figure 3-24 illustrates the coverage of a platform diagram.

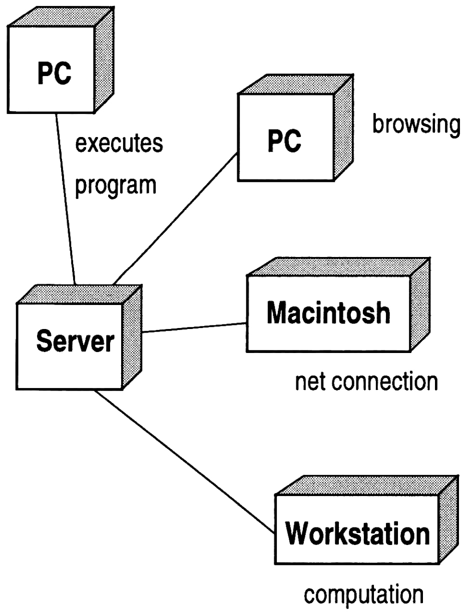


Figure 3-24: Platform Diagram Representation

Chapter 4

Related Research and Implementation

The development of distance collaboration tools is very challenging, nevertheless, there are always technological advancement and innovations that push these efforts further and faster into new territories. Currently, there are different type of collaboration tools exist in the market place a handful of such tools include the web, electronic mail, electronic discussion forum, audio, video, file transfer, chat, and document/application sharing. These tools assist users to collaborate on different levels. Web sharing and email is a form of connection establishment, but such connection is detached and it lacks feedback. On the other hand, chat and discussion forum or user group provides timely feedback, but it can hardly be classified as effective social tools, because of the lack of other human signals, such as voice and expressions. The closest replication of social contact is the video conferencing tools that carries both audio and image signals. However, it also posts some disadvantages such as creating stress on network bandwidth and being mostly limited to point-to-point connections.

The following sections are an introductory coverage of the existing distance communication products using the Internet. Those include the popular Microsoft NetMeeting, Vocaltec's Internet Conference Professional, Cornell University's CU-SeeMe, Netscape Conference, Farallon's Look@Me and the inspiring game-like 3D Worlds Chat. Table 4.1 is an evaluation summary over the presented products in various feature categories.

Table 4.1: Product Evaluation Summary

Net Conferencing Program	**Microsoft NetMeeting**	**Internet Conference Professional**	**CU-SeeMe**	**Netscape Conference**	**Look@Me**	**The Worlds Chat**
Internet Phone	yes	yes	yes	yes	yes	no
Video Conferencing	yes	no	yes	no	yes	no
Whiteboard	yes	yes	yes	yes	yes	no
Chat	yes	yes	yes	yes	yes	yes
File Transfer	yes	yes	yes	yes	yes	no
Document Application Sharing	yes	yes	no	no	yes	no
H.323 Compliant	yes	no	no	yes	no	no
# of Conferees	no limit	no limit	no limit	2	2	no limit
Casual Contact	no	no	no	no	no	no

4.1 Microsoft NetMeeting

Microsoft Netmeeting is the only product that supports all of the collaboration tools (audio, video, file transfering, chat, document/application sharing, and whiteboard). Since it is part of the popular internet explorer package, the NetMeeting interface is very easy to use. In addition, the tab-oriented interface brings in a very smooth transition between the different services NetMeeting provides.

The group conference gathering is easy, one can connect themselves to one of the Microsoft's directory servers. NetMeeting also provides filters to reduce the list of people that you see in the directory, and it let user to choose as unlisted. This unlisted feature allows hiding which has real life resemblance to physical disappearance.

Netmeeting is H.323 compliant. H.323 is a multimedia videoconferencing protocol that has become industry standard since 1995, it was approved by International Multimedia Teleconferencing Union. The audio and video quality of NetMeeting is quite good, particularly over an ISDN connection, though the video quality drops dramatically while conducting a meeting over a dial-up analog link.

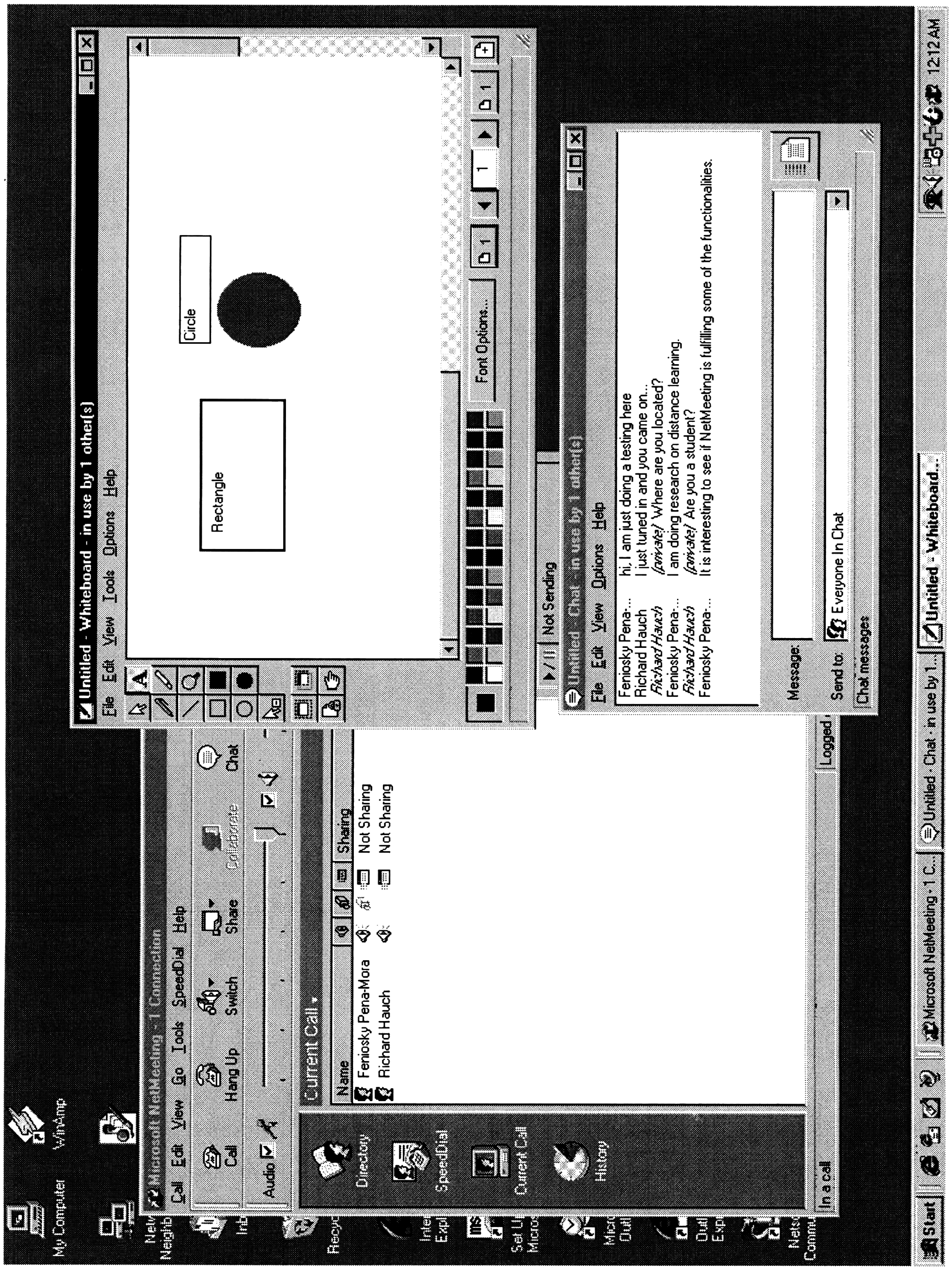
The other good feature about NetMeeting is the ability to work collaboratively on documents. One can also share an application with others even if one side does not have the application installed on his/her system. There are two modes allowed in the application sharing: view only and group effort. In view only mode, only the initiator can make changes, in the group effort mode, other users can also participate. The other good feature of NetMeeting is the whiteboard, which is a common space where people can draw, put up text and paste pieces of other information. The whiteboard is a great tool for brainstorm sessions, ideas can be expressed visually.

The shortcomings of Netmeeting come in two areas. One is that the application is not platform independent, currently it only runs on computers that have windows NT or windows 95 installed. Second, it does not provide casual contact opportunities. Each session has to be initiated and responded by involved parties (See Figure 4-1).

4.2 Internet Conference Professional

Vocaltec's Internet Conference Profession (IC Pro) offers the fastest and most sophisticated document collaboration tool. However the software does require all users to run Windows and have the shared document's native application locate on the local hard disks.

Fast document collaboration is the hallmark of this product. In addition, the package also provides audio, chat, file transfer, and whiteboard capabilities. However, the video is not included at the current version 2.0. The five modules include a well-rounded set of mark-up tools, a multiuser chat screen, and a file-transfer feature, are all tuned well enough to handle over the net activities smoothly.


Microsoft NetMeeting - 1 Connection
Call Edit View Go Tools SpeedDial Help
Call
Hang Up
Switch
Share
Collaborate
Chat
Audio
Directory
SpeedDial
Current Call
History
Current Call
In a call
Untitled - Whiteboard - in use by 1 other(s)
File Edit View Tools Options Help
Rectangle
Circle
Font Options...
Not Sending
Untitled - Chat - in use by 1 other(s)
File Edit View Options Help
Message:
Send to:
Everyone In Chat
Chat messages

Name	Sharing
Feniosky Pena-Mora	Not Sharing
Richard Hauch	Not Sharing

Feniosky Pena-...	hi, I am just doing a testing here
Richard Hauch	I just tuned in and you came on...
Richard Hauch	*[private]* Where are you located?
Feniosky Pena-...	I am doing research on distance learning.
Richard Hauch	*[private]* Are you a student?
Feniosky Pena-...	It is interesting to see if NetMeeting is fulfilling some of the functionalities.

Figure 4-1: Microsoft's NetMeeting

To convene a meeting, user can connect to one of the several Internet Conference servers, just as Microsoft's NetMeeting. After the connection, user can create a private virtual conference room, and invite others to enter his/her domain of the Internet Conference Pro.

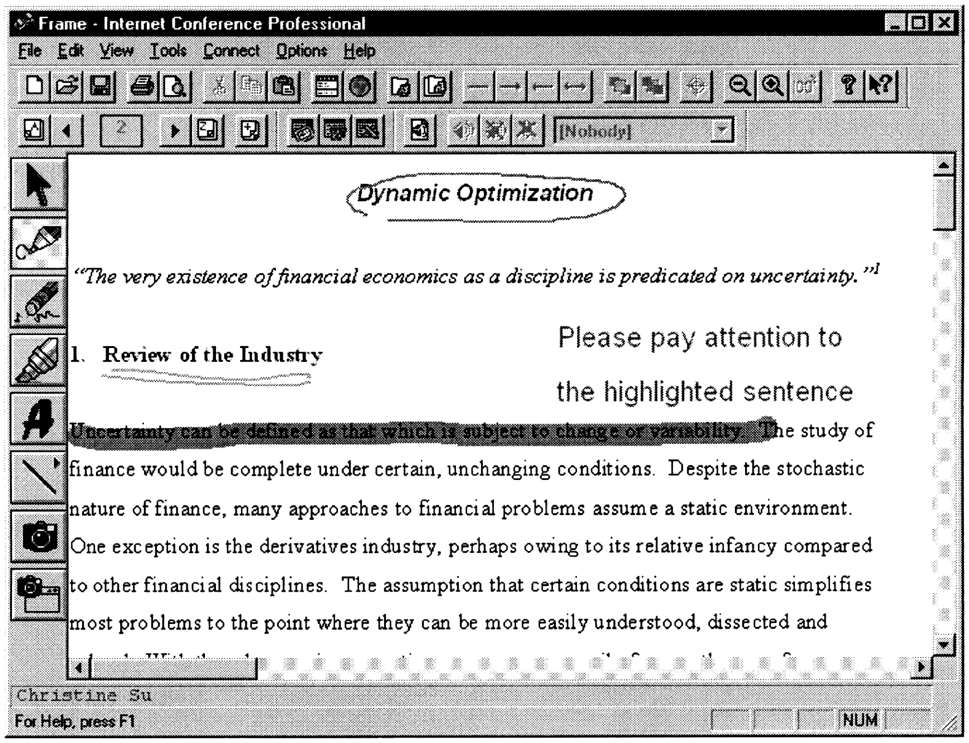
The most impressive tool of Internet Conference Pro is still its document sharing. While NetMeeting swaps bit-mapped images of the document manually, IC Pro shares OLE objects of documents that are generated by Excel or Word. Since these objects are smaller than a corresponding bitmap, changes to a document appear almost instantly on other users' screens.

In addition, IC Pro provides icons for inserting Office documents into whiteboard. A double click on the document opens the corresponding application on the local client machine, so changes can be made within. In general, IC Pro does provide the basic tool set to achieve a fulfilling distance conferencing experience. However, it does not live up to the expectation of supplying a real-life environment for collaborative learning. There are no casual interaction allowed when a user is not actively participated meetings (See Figure 4-2).

4.3 CU-SeeMe

Cornell University's development CU-SeeMe focuses on providing video and audio conferencing functionalities. As a result, it does neglect the importance of other collaboration features, such as document and application sharing. CU-SeeMe does offer some basic utilities, including a whiteboard, file-transfer service, and a text-based chat. It is also one of the only products that provide multipoint audio and videoconferencing.

Video and audio performance with CU-SeeMe are strong, equivalent to ones of the NetMeeting. In addition, instead of being limited to one-to-one video conference, CU-SeeMe grants the user the privilege to create multipoint conferences. One can put up


Frame - Internet Conference Professional
File Edit View Tools Connect Options Help
[Nobody]
Christine Su
For Help, press F1
NUM

Dynamic Optimization

"The very existence of financial economics as a discipline is predicated on uncertainty."[1]

1. Review of the Industry

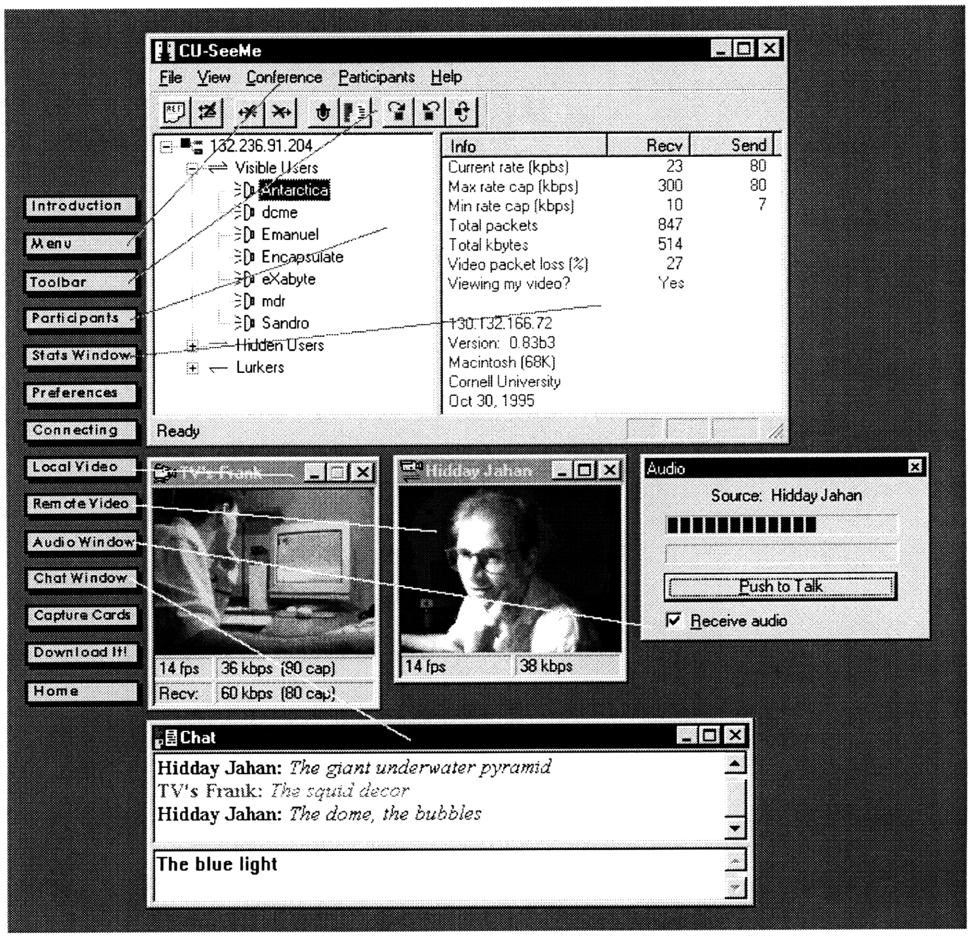
Please pay attention to the highlighted sentence

Uncertainty can be defined as that which is subject to change or variability. The study of finance would be complete under certain, unchanging conditions. Despite the stochastic nature of finance, many approaches to financial problems assume a static environment. One exception is the derivatives industry, perhaps owing to its relative infancy compared to other financial disciplines. The assumption that certain conditions are static simplifies most problems to the point where they can be more easily understood, dissected and

Figure 4-2: Vocaltec's Internet Conference Professional

to 12 video windows on the screen simultaneously, each displaying the image stream of a different conferee. The performance will degrade with addition of conferencing windows, making this feature only suitable for conferencing over high speed networks.

While CU-SeeMe's video and audio conferencing capabilities are impressive, it is not quite H.323 compliant yet. It is also platform dependent because it can only be run on windows system. CU-SeeMe's primary collaborative tool is its whiteboard. Even though its whiteboard includes some extraordinary features like screen shot and image display, it is still falling short on allowing real-time document sharing and editing. Similar to NetMeeting and IC Pro, CU-SeeMe does not provide any casual collaboration tools, which proves that implementations of any representation of the real-life interactions are very difficult (See Figure 4-3).



Info	Recv	Send
Current rate (kpbs)	23	80
Max rate cap (kbps)	300	80
Min rate cap (kbps)	10	7
Total packets	847	
Total kbytes	514	
Video packet loss (%)	27	
Viewing my video?	Yes	

130.132.166.72
Version: 0.83b3
Macintosh (68K)
Cornell University
Oct 30, 1995

14 fps 36 kbps (80 cap)
Recv: 60 kbps (80 cap)

14 fps 38 kbps

Source: Hidday Jahan

Hidday Jahan: *The giant underwater pyramid*
TV's Frank: *The squid decor*
Hidday Jahan: *The dome, the bubbles*

The blue light

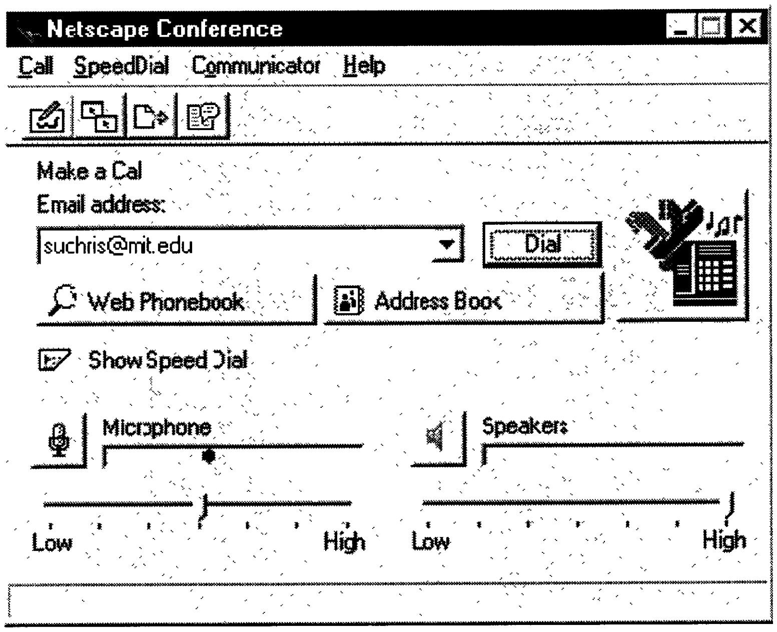
Figure 4-3: Connel University's CU-SeeMe

4.4 Netscape Conference

Netscape Conference is part of the package within the Netscape's Communicator suite. Its greatest strength is its multiplatform availability, because it is browser based. To convene a meeting, the user is linked up to a lookup server which provides email address listing. Once connected, user can chat, talk, pull up whiteboard, or transmit files.

Although Netscape Conference delivers superb sound quality, it short falls on the inabilities to share documents or provide video. Its collaborator toolset is also weaker than either NetMeeting and IC Pro. The omission of dynamic document editing does introduce inconveniences for users. On the other hand, Conference is the only product that offers a feature called Collaborative Web Browsing. This feature allow a meeting initiator to lead other participants to browse his/her web presentation automatically. It is similar to a PowerPoint slide show or WebPresenter.

The Conference has some nice touches, but they are not enough to overcome the fact that the service is limited to one-to-one conferencing. Adding to the shortcoming list, Conference also does not provide environment to experience casual contact or unplanned social interactions. To sum up, Netscape Conference's toolset is inadequate compared to NetMeeting and IC Pro's (See Figure 4-4).



suchris@mit.edu

Figure 4-4: Netscape Conference

4.5 Look@Me

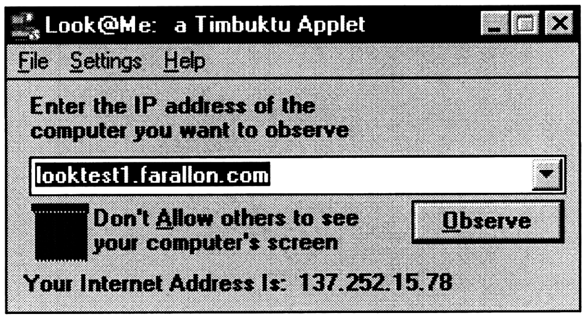
Look@Me is a free mini-application developed by Farallon that allows user to observe one another's screen anywhere in the world, in real time, over the Internet. Based on the Netopia's Timbuktu Pro for Windows, Look@Me showcases Timbuktu Pro's Observe screen-sharing feature in a form that has been optimized for use over the Internet.

Users choose Look@Me for personal Internet communication - s/he can collaborate in real time with any other Internet user anywhere in the world. Though Look@Me was developed from Timbuktu Pro, it offers more features than its predecessor. Look@Me provides four services: Control, Observe, Send, and Exchange. Send allows user to create FlashNotes and attach files. Exchange allows users to treat a remote computer's disk drives as if they were attached to their PC.

Both Timbuktu Pro and Look@Me users can connect with any other either through Windows or Macintosh. It supports TCP/IP and IPX. Look@Me also ships with Shiva remote node software, allowing you to connect to any PPP server and use Timbuktu Pro over the remote network link.

One advantage of Timbuktu Pro over Look@Me is that it has full-featured security with registered user and password support. Look@Me can be turned on and off to allow or deny access - it does not provide sophisticated security schemes. Timbuktu Pro has both personal and shared address books that allow the user to stored frequently used connections. Look@Me only keeps a record of the last four TCP/IP connections.

In summary, with Look@Me one can edit documents, go over presentations, review graphics, and provide *just-in-time-training* and support. Out of all of the products presented so far, Look@Me adds limited value to solve for the distance education problem. It is a sophisticated document sharing and collaborative tool, but it lacks the innovations in bringing conveniences and ease-of-use in the product. In addition, it falls short on providing a casual interaction tool knit to the users.



Enter the IP address of the computer you want to observe

looktest1.farallon.com

Don't Allow others to see your computer's screen

Your Internet Address Is: 137.252.15.78

Figure 4-5: Farallon's Look@Me

4.6 Worlds Chat

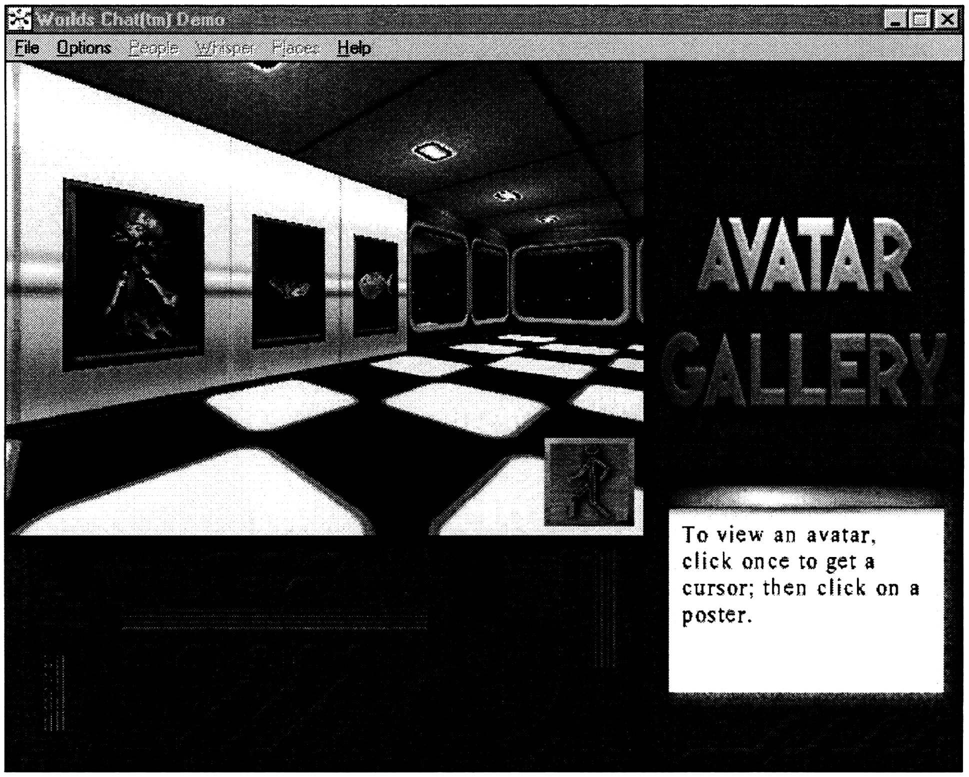
Worlds Chat is perhaps the weirdest of all avatar-based 3D virtual reality chats, Worlds Chat is a 3D social environment where user can explore individual platforms and rooms on a space station.

Once you log on to Worlds Chat, you can choose your personal avatar from a huge gallery of characters. One might find the environment easy to navigate if they have previous game experience. As you bump into doors, you melt through them and into a different room with a different feeling. You can also teleport to different parts of the station, emerging from something that looks like a cross between a volcano and a Star Trek transporter.

Other avatars have text handles to identify them: the Count, for example, was a friendly vampire who would show users around and helped them figure out how some of the features worked. A "whisper" feature–the equivalent of private chatlets–lets you carry on one-on-ones with any other person in the scene. In fact, since everyone uses whisper almost exclusively, you may think you've gone deaf when you first enter. It's like being an average person in a space station full of telepaths.

Because it's easy to get lost in the space station, Worlds Chat's help function was especially nice. You can display a rotating selection of tricks, shortcuts, and fun places to go, and leave the help window open while you're moving around. Worlds

Chat is fun and strange, and comparatively easy–at least, as far as avatar-based chat goes. However, like every other chat application I have encountered, it is more for entertainment than for learning. The main purpose of the application is to enhance the cyber social environment and let people experience what they can't in real life. It seems to me that anything that is meaningful or educational seems to be beyond the scope of Worlds Chat's creators.



To view an avatar, click once to get a cursor; then click on a poster.

Figure 4-6: Worlds Chat's 3D User Interface

Chapter 5

Implementation: Requirement Analysis

The goal of this thesis is to present a workable approach to build a collaborative distance learning system. There are many implementation possibilities for such a problem, however it is important to define and focus on the necessary features before moving forward into the software design phase. The following section will give a complete analysis of the distributed collaborative learning problem, and further specifying the necessary functionalities of designing the *Cliq!* system.

In this section, you will observe the connections between requirement analysis and software design. The close knit relationship between these two phases are apparent, though there are not any implementation decisions made during the requirement analysis phase, it helps designer to narrow down the system considerations to several specific areas. It is crucial to know what the clients' demands and expectations are, and deliver those needs accordingly.

5.1 Requirement Analysis

The requirement analysis phase is conducted mainly by the requirement analysts Humberto Chavez and Simonetta Rodriguez. The client, in this case, our instructor Feniosky Peña-Mora comes to our team for software solutions to a distance communication problem. The goal of the requirement analysis phase is to identify the client's needs and transform those needs and ideas to concrete forms that can be illustrated by words or diagrams. You shall find the complete requirement analysis document provided below.

5.2 Introduction

The terms "distance education" or "distance learning" have been applied interchangeably by many different researchers to a great variety of programs, providers, audiences, and media. Its hallmarks are the separation of teacher and learner in space and/or time [23], the volitional control of learning by the student rather than the distant instructor [20], and noncontiguous communication between student and teacher, mediated by print or some form of technology [13] [21].

Today, political and public interest in distance education is especially high in areas where the student population is widely distributed. Each region has developed its own form of distance education in accordance with local resources, target audience, and philosophy of the organization which provide the instruction. Many institutions, both public and private, offer university courses for self-motivated individuals through independent study programs. Students work on their own, with supplied course materials, printed based media and postal communication, some form of teleconferencing and/or electronic networking, and learner support from tutors and mentors via telephone or E-mail.

Although technology is an integral part of the distance education, any successful program must focus on the instructional needs of the students, rather than on the technology itself. It is essential to consider their ages, cultural and socioeconomic backgrounds, interests and experiences, educational levels, and familiarity with distance education methods and delivery systems [26]. Thus distance education is not an isolated phenomenon; it is affected by the political, social, financial, and technological factors in its environment.

Distance learning allows students to hear and perhaps see teachers, as well as allowing teachers to react to their students' comments and questions. Moreover, virtual learning communities can be formed, in which students and researchers throughout the world who are part of the same class or study group can contact one another at any time of the day or night to share observations, information, and expertise with one another. If distance learning is possible and desirable, then the home, the office, or the hotel room became the classroom. The learner and the teacher study together on schedules which are convenient to both.

Successful distance education system involve interactivity between teacher and students, between students and the learning environment, and among students themselves, as well as active learning in the classroom. Interactivity takes many forms; it is not just limited to audio and video, nor solely to teacher-student interactions. It represents the connectivity the students feel with the distance teacher, the local teacher, aides, and facilitators, and their peers.

The instructional development process for distance education, consist of the customary stages of design, development, evaluation, and revision. In designing effective distance instruction, one must consider not only the goals, needs, and characteristics of teachers and students, but also content requirements and technical constraints.

5.3 Implication of Current Problems

Current techniques and technologies for conducting distance learning and distributed project collaboration include significant deficiencies. These include:

1. Severely limited or non-existent **social interaction**[1] between participants. The client perceives that social interaction within classrooms and collaborating groups allows:

   (a) lasting bonds" among the participants. Lasting bonds are viewed as essential for effective learning and collaborative environments.

   (b) participants to " interpreting the thinking" of other participants. The client perceives that groups work better when group participants can predict some of the future behaviors of other participants.

2. Severely limited or non-existent **social feedback**[2] while speaking, lecturing, or attempting to interact. Social feedback, as a component of social interaction, is viewed as essential for appropriate delivery of content, as well as for assurance that intended communications have been established. Examples of social feedback include:

   (a) facial expressions

   (b) hand movements

   (c) orientation of the head

   (d) focal point of the eyes

3. Non-existent opportunities for unplanned encounters or **casual social interaction**[3] in informal settings unrelated to the structured sessions of the project or class.

[1]Social interaction is a *technical term* in the field of environmental psychology, referring distinctly to the interaction of one or more human beings with other human beings in a specific setting. The term is used here in this technical sense, to distinguish from other kinds of interaction, such as those between human beings and machines. This usage conforms closely to what is implied by the client's use of the single word interaction.

[2]Social feedback is used here to refer to those components of social interaction that permit a speaker to determine the state of the listener. This allows a distinction between feedback information perceived by human beings about other human beings, and machine feedback mechanisms. In other words, machines can provide feedback, which may or may not provide human beings with social feedback about other human beings.

[3]Casual social interaction is used here to refer to such unplanned, spontaneous social interaction occurring outside of planned lectures, sessions and meetings.

The client views casual social interactions among participants as significant contributors to effective learning and collaborative environments, because they allow observation of unrehearsed behaviors. This facilitates learning about the other participants than can not be learned without such unplanned encounters.

5.4 Broad Goals

1. Re-create the "campus experience" through electronic means for distance learning situations.
2. Allow individuals to express themselves without the constraints of the (machine) environment.

5.5 Detailed Analysis

5.5.1 Initial Statement of the Proposal

The system must provide significant improvement for social interaction beyond what is offered by existing collaboration and distance learning systems. Components of social interaction that must be included:

1. "Collaborative awareness" of other participants. To provide collaborative awareness, the system must transmit social feedback information, which must include, at minimum, the following:
   (a) the presence of other participants
   (b) some of the behaviors of other participants
   (c) the attention state of other participants
2. Possibilities for "casual social interaction" among participants, outside of planned sessions and structured meetings of the project.

3. Possibilities for "personal expression" with freedom from the constraints of the technological environment.

5.5.2 Comparative Analysis

Before detailed analysis of the requirements stated above, a brief comparative analysis is useful. What functionality provided by existing products satisfy some or all of the client's requirements? Please refer the previous chapter for more comparative analysis.

The most mature of this product class, called groupware, is Lotus Notes. One representation of the functionality provided by Lotus Notes and similar products is shown in Figure 5-1.

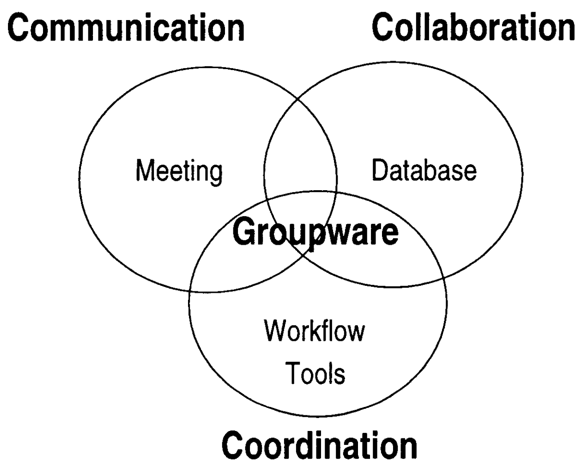


Figure 5-1: Groupware Software Product Domain, Adapted from lecture note of 15.564[5]

Note that each of the functions of integrated groupware products can be provided by separate software products. The advantage of groupware products is that integration is already accomplished. Integration can also be a disadvantage, since better tools for each component may be available, and upgrading components separately may be desirable. In relation to the previous representation of current groupware products, the system to be developed must offer an additional component, an enhancement beyond communication, collaboration, and coordination. This additional component is Social Interaction

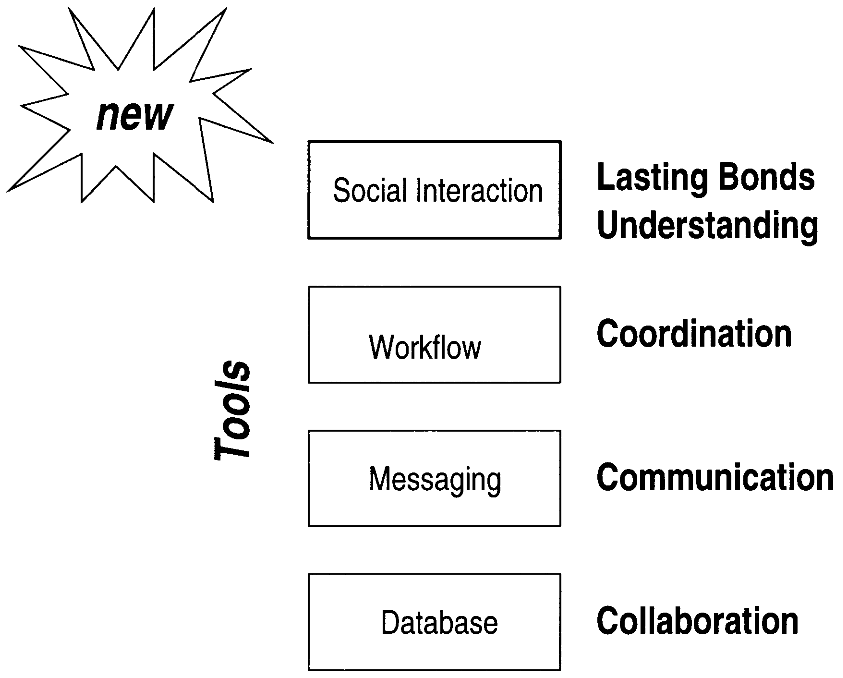


Figure 5-2: Importance of Social Interaction

However, note also that every component in this model is a form of communication between human beings.

5.5.3 Social Interaction Analysis

Social interaction, as stated by the client, is comprised of the three elements: 1. social feedback, 2. casual contact, 3. personal expression. It is clear that the first and last of these are directly related to each other, because portions of another person's expressions - speech, behavior, and so on – provide the signals interpreted as feedback. The client's primary goal, to provide social interaction, can be accomplished in two forms: during planned encounters among participants, and during unplanned encounters.

Examples of planned encounters include scheduled events, lectures, works sessions, and project meetings. Examples of unplanned encounters include chance meetings among participants, what we call here casual contact. Both forms are composed of couples: personal expression and social feedback, represented by the interlocked shapes in the diagram.

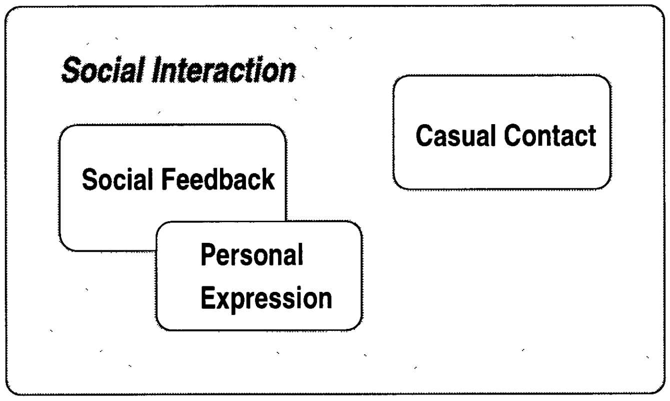


Figure 5-3: Social Interaction Components

It is important to note the tight coupling of social feedback and personal expression in both planned and casual forms of social interaction. People express themselves verbally and non-verbally, through activity and through inactivity, voluntarily and involuntarily. Some portion of the total range of a person's expressive behaviors, provides the cues that others use to interpret the person's state. We call this social feedback The system must transmit some representation of personal expression, to those who wish to obtain social feedback.

5.5.4 Personal Expression Analysis

To analyze the concept of personal expression as stated by the client, a representation called a **black-box-human**[6] is introduced here. Each participant is represented as a

[6]It is worth noting that the fields of sociology, psychology, political science, and anthropology, to name four out of many, are each, in part, devoted to understanding the issues and dynamics of

black-box-human providing output, e.g. emitting signals.

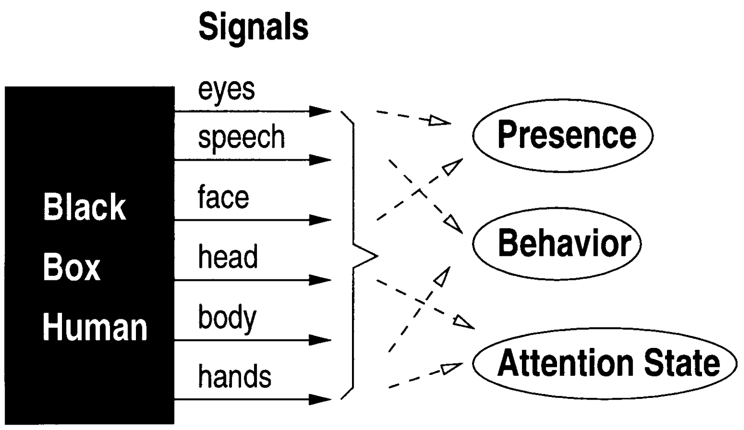


Figure 5-4: Personal Expression Components

Six readily identified signals emitted by a black-box- human operating in a social setting:

- movement, lack of movement and focal point of the eyes
- verbal expression or its absence , e.g. speech
- movement of facial muscles or lack of movement in the face
- position, orientation, and movement of the head
- position, orientation, and movement of the body (so-called body language)
- position, orientation, and movement of the hands

Note that this is not an exhaustive list, additional signals can be identified very easily. Additional signals found to be essential to the purposes of the project should be added to the list. Taking a cue from the signal labeled "body", commonly called

personal expression in social settings. The **black box human** representation enables a beginning for this analysis, undertaken in order to develop a software product, rather than to engage in philosophical debate.

"body language", a little reflection leads to the conclusion that each of these signals is a language used by human beings to communicate in social settings. Thus the six identified signals are transmitted in six "native languages". In our representation, these six signals, emitted by the black-box-human in their six respective native languages, transmit the components of social feedback identified by the client: presence, behaviors, and attention state. Further analysis of personal expression calls for the following:

1. Identify key forms or instances of each signal in its native language
2. Of these signal forms or instances, and combinations of them, identify which are essential for transmitting presence, behaviors, and attention state.
3. Identify how additional or alternate languages, suitable for transmission through computer systems, will be developed for some or all of the native black-box-human signal languages. It is crucial to note, that new languages for representing these signals, must be tested for their ability to be interpreted by the intended market of system participants, in the social feedback stage.
4. Identify how to translate the essential signals or combination of signals into the new languages.

5.5.5 Personal Expression Summary

For the purposes of this project, the personal expression construct is simplified as follows:

1. The signals that must be translated by the system are: body, head, face, and hand, and combinations specified below. Speech, while also a required signal, is readily transmitted by existing technologies.
2. The Body signal is essential for transmission of all three requirements. Body signals that must be translated and transmitted by the system:

   (a) Each participant's bodily presence during planned sessions, continuously throughout planned sessions.
   (b) Intermittent (non-continuous) bodily presence outside of planned sessions, indicating availability for casual contact.
   (c) Sitting
   (d) Standing
   (e) Shoulders up
   (f) Shoulders down
   (g) Walking away

3. The Head signal is essential for transmission of all three requirements. Head signals that must be translated and transmitted by the system:
   (a) Orientation (which way facing)
   (b) Upright
   (c) Tilted
   (d) Nod
   (e) Shake

4. The Face signal is essential for transmission of all three requirements. Face signals that must be translated and transmitted by the system:
   (a) Smile
   (b) Laugh
   (c) Frown
   (d) Serious
   (e) Yawn

5. The Hand signal is essential for transmission of all three requirements. Hand signals, two for each participant, modulated individually, that must be translated and transmitted by the system:
    (a) Raised
    (b) Point
    (c) Fist
    (d) Count
    (e) Held up to head
    (f) Palm up
    (g) Palm down
    (h) Wave

6. Certain Combo signals (combinations of the above) are required:
    (a) Excuse Me signal (wish to speak): one hand raised
    (b) Puzzled signal: body shoulders up, head tilted, both hands palm up
    (c) Demand signal: face serious, hand(s) fist
    (d) Bored signal: head tilted, face yawn, hand held up to head
    (e) Leaving Now signal: body walking away, hand wave
    (f) Negative Attention signal, no voluntary signal during specified period of time (see summary: social feedback): head tilted, face away

5.5.6 Social Feedback Analysis

With personal expression represented as black-box-humans emitting signals in native signal languages, social feedback may be represented as an interested party's receipt and interpretation of those signals. While each person in a social setting continuously emits a variety of signals, an interested party receives some of these signals at various times. These reception times are more or less within the interested party's control, especially in a group setting.

In a group setting, the interested party must interpret the same signals at two (minimum) levels, individual level interpretation and group level interpretation.

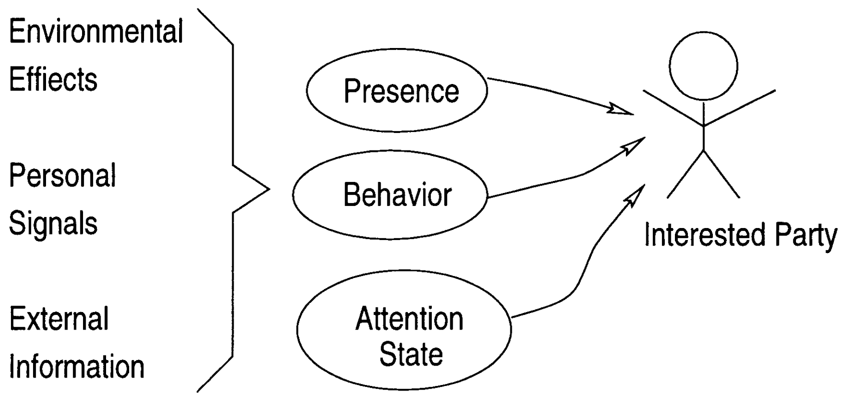


Figure 5-5: Social Feedback Components

Example: A lecturing professor often faces a blackboard or projected image rather than the audience of students. Professors usually choose to periodically look at the students in the room or hall, to see if they are paying attention. If a professor receives the following signals from six students:

1. head tilted forward

2. eyes closed

3. body still

4. snoring sounds

he or she is likely to interpret the following for the six students:

1. presence: positive – they are present

2. behaviors: they are asleep
3. attention state: negative to the extreme

The professor, in addition, may interpret the same signals at another level, that is, in regard to the whole group of students. Interpretations at the group level and the individual level may not, necessarily, agree or correspond in any direct manner.

For the example above, the professor could interpret the following for the group:

1. presence: negative – six students represent only 10enrolled in the class;
2. behaviors: something happened today that kept 90from attending class;
3. attention state: either the lecture is far too boring, or something else is happening about which more information is needed.

Further analysis of social feedback calls for the following:

1. Identify the constituent factors for interpretation of received personal expression signals.
2. Identify the critical factors necessary for interpretation of presence, behaviors, and attention state in the settings in which the proposed system is expected to operate.
3. Identify how to test interpretations by different parties at different times.

5.5.7 Social Feedback Summary

For the purposes of this project, the social feedback construct is simplified as follows:

- The signals specified previously – body, head, face, hand, and combos – along with speech, which may be transmitted by external technologies, are assumed to be sufficient for an interested party to interpret and perceive presence, behaviors and attention state.

- Most signals are assumed to the result of voluntary actions by participants.
- Presence should be interpreted when the body signal operates.
- Behaviors should be interpreted through the operation of any voluntary signal.
- Attention state for each participant should be interpreted as follows:
  - Positive Attention state: voluntary signals occurring.
  - Negative Attention state: no voluntary signal within a specified period of time.

5.5.8 Casual Contact Analysis

With the preceding representations of personal expression and social feedback, casual contact may represented as a "when" issue rather than a "what" issue. Since the components of social interaction are personal expression and social feedback, the fundamental criterion for the existence of casual contact, is determined by when social interaction takes place: during planned events or sessions, or during unplanned events or encounters. Various forms of black-box-human signals can and will be emitted during both planned and unplanned events, and must be interpreted by an interested party to obtain social feedback.

To move beyond the fundamental criterion of casual contact, (that it is unplanned social interaction), we examine how casual contact occurs in "real life" (real time contiguous space, which will be noted RTCS), and note important features the client considers crucial to preserve:

1. RTCS casual contact events occur essentially randomly. This follows from the fact that they are unplanned. Patterns in RTCS casual contact do appear, due to factors such as personal schedules, physical proximity of living space or office quarters, preferences for times or places to eat, and so on. Individuals find that they "run into" certain other individuals more often than others, in certain

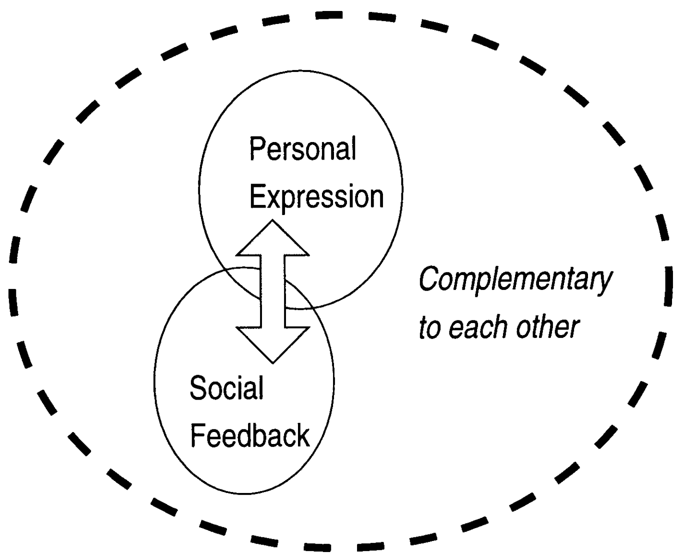


Figure 5-6: Components of Social Interaction

places and/or at certain times. Yet these patterns do not guarantee contact; there is a large element of chance. The client considers the random quality of RTCS casual contact to be a crucial component to preserve in a computerized social interaction system which provides opportunities for casual contact.

2. While RTCS casual contact events occur randomly, individuals are able to exercise various levels of personal control over them. For example, an individual who knows that an encounter with Professor Peña-Mora is more likely to occur on the stretch of hallway near his office, may choose to either a) avoid that hallway, or b) to use it more frequently, with the choice based on desire for casual contact with the professor. Professor Peña-Mora, on the other hand, like all human beings, routinely uses a variety of personal expression signals to indicate his willingness to engage in social interaction at any given moment. He may, for example, state "See me later, I have a meeting right now." Or he may walk very fast with head down, which most people would interpret as "do not disturb". On the other hand, if one encountered the professor (this is hypo-

thetical) in a relaxed pose, apparently lounging in the hall outside of his office, most people would assume that he is open to casual contact in that moment. The client considers the personal control features of RTCS casual contact to be a crucial component to preserve in a computerized social interaction system which provides opportunities for casual contact.

3. RTCS casual contact events may be viewed as primarily related to spaces, activities, and broadcasts. While all RTCS events occur in the same space (by definition), some are more directly related to spatial and locational issues, for example, the hallway contact mentioned above. While all RTCS events occur at the same time (by definition), some are more directly related to activities rather than spaces, for example running into someone when faxing, at any of several fax machines in the vicinity. Another example of activity-related casual contact is running into someone when eating, at any of several eating locations that one frequents. Broadcast casual contact may be viewed as the result of broadcasting one's availability for casual contact.

4. Issues involving definitions about what is private and what is public must be considered for casual contact provided as part of a computerized social interaction system. We note that public/private issues are also involved in the RTCS case and that some portion of the legal system in most countries revolves around this issue. The ability to exercise personal control over casual contact may be viewed as essential to avoiding problems in this area.

5.5.9 Casual Contact Summary

Mechanisms for casual contact among participants must be provided by the system. These mechanisms should include the following characteristics:

1. Casual contact events must include a random quality or a sense of randomness. One way to do this might be to define classes of atomic events that serve as

markers for the possibility of casual contact events. An atomic event could be something like two participants retrieving mail from a post office server at the same time.

2. Participants can choose to signal their availability for casual contact. A subset of the personal expression signals may be used, or a special signal may be developed for this purpose.

3. Participants can choose to actively signal availability on a once-at-a-time basis, on a timed basis, or automatically in response to specific activities on various computers. For example, one participant may attach casual contact activation software to certain applications used frequently, such as a mail program, the Win 95 Recycling Bin, a browser, or a spreadsheet application, some combination of these, or others. Another participant may choose to signal availability every day at a certain time or virtual location. Another participant may choose to signal availability after a certain number of keystrokes within a specific time period, equivalent to the rest breaks recommended for carpal tunnel syndrome.

4. Casual contact activation signals and atomic event signals must be transmitted to registered participants of the system.

5. A method for receiving and displaying casual contact activation signals must be provided.

6. A method of responding to a casual contact activation signals must be provided.

7. A method of accepting a response must be provided.

8. A range of options should be available for developing an instance of social interaction after a response has been accepted. For example, the two (or more) participants involved in a social interaction instance, may choose to transmit signals through the social interaction system or tool, while also using external tools to engage in chat, telephony, and/or whiteboard interactivity sessions.

5.5.10 Social Interaction: Further Representation

After the preceding analyses and representations, examining social interaction in yet more detail at this point is instructive. Several conclusions can be drawn, and debated.

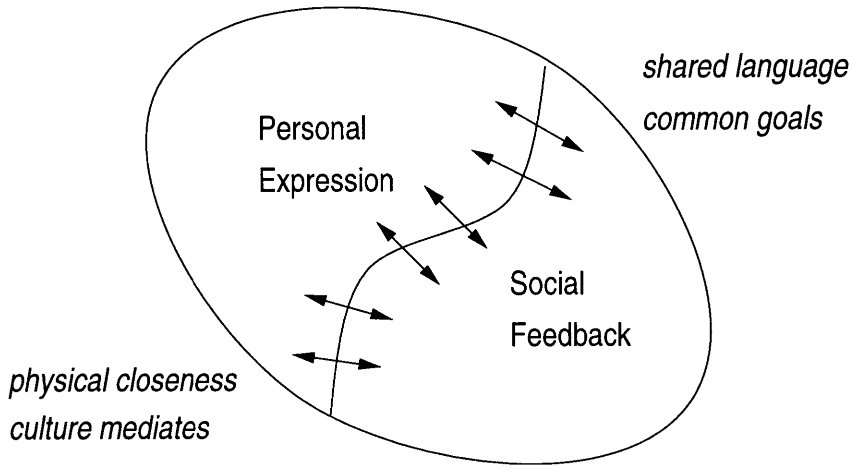


Figure 5-7: Social Interaction Components in the Ying-Yang Model

1. The components of social interaction are personal expression and social feedback.
2. The languages of emitted signals must be shared between black-box-humans and interested parties, for social feedback to occur.
3. As a result, these two components are tightly integrated – they do not occur in isolation from each other;
4. For a particular integrated system, that is, for a unique set black-box-human, shared signal language, interested party, the system holds across space and time.
5. Culture, whether shared or differing between participants, will affect both personal expression and social feedback, and thus social interaction systems.

5.5.11 A Social Interaction Tool

Integration of the client's stated requirements into a view of the system to be developed as an extension of groupware functions, is now possible. The problem stated by the client resolves to providing a tool that enables social interaction, defined as a specialized class of communication between human beings, to be transmitted via computers. This social interaction tool should handle social interaction in both planned and unplanned interaction sessions.

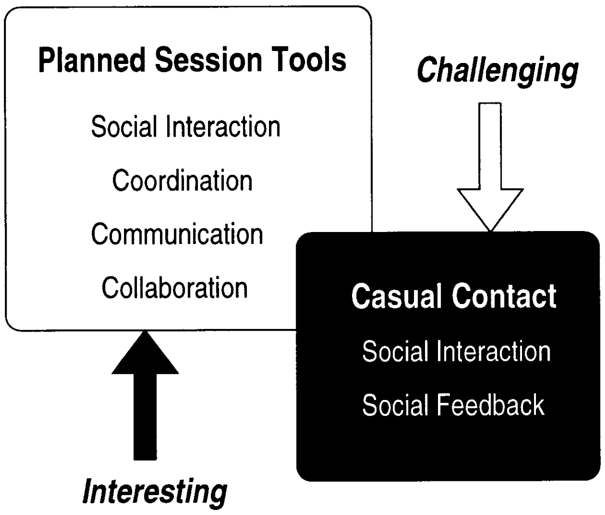


Figure 5-8: Comparison between Social and Casual Interaction Tool

The social interaction tool can provide social interaction by translating some or all of human personal expression signals, into alternative languages suitable for computer transmission, but still interpretable by human beings. It enables the basic components of social interaction, which are personal expression and social feedback.

When engaged in planned sessions, the social interaction tool augments the communication functions provided by existing tools. It can be integrated into a complete system or it can be implemented as a separate system. This is a design decision.

When engaged in unplanned sessions, the social interaction tool can stand alone. By the very nature of combining unplanned sessions and social interaction, the social interaction tool permits a degree of shared understanding not currently possible with existing tools.

Chapter 6

Implementation: Design

Software Design phase is comprised of many small steps, each can be viewed as a milestone moving closer to the far reached goal. By now, you should be familiar with object-oriented design approaches and the UML notation presented in Chapter 3. Such knowledge will help you to understand the following collection of design documents clearly.

6.1 Scenario Description

The initial step in the design phase is to come up with a complete list of scenario descriptions. The descriptions are blueprints of the system. They cover what the system is responsible for at execution, how the data flow and event generation will evolve, and most importantly, what are the dynamics between human and machines at run time. The scenario description document is a first attempt to make implementation decisions, it paints imagery of the looks and feels of the system.

Adjustment to Different Terminology

There are a lot of university related terms used in the documents that are presented in this chapter. However, such a list of naming information can vary depends on the users base of the system, if the system is used in a corporate environment, the term instructor/lecturer can be replaced by presenter, student can be replaced by meeting participant, class can be replaced by discussion sessions or presentation. The other school specific terms such as classroom can be replaced with meeting room. The terminology used here is suitable for universities, however the terms can be easily expanded to more general basis which will satisfy different needs.

Basic Scenarios

The following were five basic scenarios that are defined, within each one of them, we had to consider the answers to some questions related to the implementation:

1. The establishment of any user in the system

   How does the system know who is allowed to use the application?

   How does the system know which user has which type of privilege?

   How do the users identify themselves to the system?

2. The establishment of an virtual environment

   Should we use static environment that is hard coded in the system?

   How does the system configure the environment?

   Does the environment contain rooms, places, and objects?

3. Accessing the virtual environment

   How do users enter the virtual environment that was established?

   How do users move from one place to another?

   What constraints are there in the room or any other areas?

   Are all users equal, do they follow the same rule and have the same privileges in the room?

4. Interactions in the virtual environment

   Who can speak to who in the meeting room?

What can the user control?

What information should be available?

5. Casual Interaction

   How do users enter the casual interaction?

   What are the limitations of user controls?

User Registration

Before any users can use the application, they must register to the system, this can be done in two ways. One way is to establish the account by sending in the required information about the user to the server in a specific format from the client computer. The other way will be having the system administrator to establish the user account at the server.

An user profile or identity is created in the system either by the administrator or by the user. If it is completed by the user, the user will submit a list of information about himself directly to the server through a remote connection or an email. If the process is done by the administrator, he could enter those information for the user at the server side.

- The user will provide the following information

  Name

  Email address/System username

  Images or VRML files of the person in different expression

  Web pages or Public directory

  Student, lecturer, or teaching assistant (TA)

  Registered classes, classes that is teaching, or classes that is tutoring

- User can be identified as a person in general, the student/lecturer/teaching assistant status is entirely environment dependent, meaning a person is a student in room for class A, but a TA in another room for class B, since there is no need to have special privileges for anyone in a casual environment. The user can also configure his default system setting. User can also change their personal information stored at the server by request a modification to his/her profile.

- The server will keep a list of users that are allowed to use the system and it handles all the transactions of changes made. In addition, the server will function as the processes scheduler for the system.
- One extension of this process is user can change their personal information stored at the server anytime by request a modification to his profile.

Replay

The user will have the option to log messages exchanged between himself and others, commands or requests made to change the avatar expressions, and the contents that were displayed on the shared blackboard/whiteboard. These information would be saved as text file on the client computer. When the user needs to replay the past scene, the server will regenerate the avatar behavior by going through the log file. The replay feature can be configured as a default setting, such that once the user starts to use the Cliq! application, the recording will run automatically. The following is a list of information that can be provided by the log file:

- The area that the user is in.
- List of users that are in the same area.
- Chat messages that are displayed on the user's desktop.
- Request that are made to change avatars' expressions.
- All other commands that the user make to change room, write on whiteboard, and post notes.

One suggestion to enhance the replay function is to have a pre-set recording that is similar to the VCR's timed record feature. A user can request a log file of a lecture that is conducted in a particular meeting room at a specific time.

Establishing Virtual Environment

The system can have a pre-defined or a standard environment, but it is preferred to have a configurable setting depends on the users' needs. This environment can include, for example, a meeting room for class A at one time slot and class B for another time slot, a common hallway, a library, and a lobby. Once the basic environments are established, it can extend to include bathroom, coffee shop, or private offices. In addition, every user can create their own private room which will be described below.

There are three categories of areas, the public area, special public area, and private area. The definitions of each category are give below:

Public Area e.g. lobby, hallway, bathroom

In this type of area, every user has the same level of control. There are no owners for the area, and limited number of user presences is optional. There is no limitation of whom one can speak to, both one-to-one and one-to-many conversations are allowed, and these two communication modes can be selected by the users that are presented in the area.

Special Public Area e.g. library, meeting room

There are some constraints in this special area, for example, in the library, users can only speak one-to-one. In addition, they can post notes on whiteboard and send chat messages to others. There will be a different set of constraints in a meeting room, such as timed ownership set by lecturer, rule establishment by the owner of the room, and other general access controls

Private Area e.g. lecturer's office, TA's office, user's room

This type of area can have permanent ownership. The area can be locked and unlocked by the owner. The owner can also established a set of rules that other users must follow when present in the area, otherwise the private area is very much like the special public area.

Create a room who can and how

In general, a user can create a new room as one of the three areas defined above, and he/she can claim ownership of the room, set capacity of his room, set mode of conversation is allowed in his room, or set access right to his room.

Access rules for all environments

There are other rules applied to room usage, such as a user can not enter a locked room. However, anyone can leave an locked or unlocked room freely. When the room is in full capacity, no additional user can enter, however this must not be applied to most of the public areas. A user can not be present in more than one area, and all users are given the options to find out who is in what areas with exceptions to the areas that are locked. As a side note, the document that are shared in the lectures are not accessed through our system, Netscape or Internet explorer can support this service.

Accessing the Virtual Environment

We present two type of user accesses here, one for the student/TA, and the other for the lecturer:

1. The user will log in to the system by providing his/her username and password, once the system positively identified the person, it will provide the right profile that the person had registered earlier.

2. If the user chooses to bring up the Cliq! application, an avatar representing the user will appear in the lobby area

3. The user can request to move into the meeting room, by issuing a command *ChangeRoom* from selecting an item from the menu bar.

4. If the user has student or teaching assistant status, the system checks whether he is not registered for the class, and if the area is locked or over filled, if so, the request is rejected, else the avatar is moved to the requested room.

5. If the user is a lecturer, the system checks whether he has the ownership for this time slot, if so, moves the avatar in the room. If the check fails, the system will continue with the processing steps for regular users described above.

6. The lecturer can request for change of meeting room settings, such as
   allow/disallow students/TAs speak to all people in the room
   allow/disallow listeners (students/TAs/lecturers that are not on the class list)
   allow/disallow others to write or post documents on the blackboard
   allow/disallow others to enter the meeting room after a specific time

Any temporary ownerships will terminate upon its expiration, and all the rules and settings that the owner establish will expire as well. There should be a default setting in every area (public, special public or private) to start with. In addition, the owner can pass out key (tokens) to others to access the locked area when he/she is not present. Users can learn about the rules within a class upon entering, such as getting a list of dos and donts.

The distinction between an instructor and a student is only through the status of the *user* class. It is not necessary to have different roles in the system. An instructor can be defined as a *super user* for a certain period of time, in a certain room. In public area, all users are equal, and there are no difference between an instructor, a student or a teaching assistant.

Virtual Environment Interaction

A typical scenario of the meeting room interaction is:

- The presenter enters the meeting room, and changes the meeting room settings. He disallows other users to interrupt his presentation, and disallows listeners. The system then moves all non-class users to a public area, and rejects all other users' requests to *Speak to All* commands.

- The presenter starts to deliver the lecture, if a student have a question, he sends in a command by changing his avatar to a different image, such as a representation with raised hand.

- The presenter observes the request, and allows the student to speak up. He then proceeds to answer the question. The presenter can also allow group discussion, when he changes the meeting room setting to allow everyone in the room to speak up.

- Other users can speak one-to-one, post notes and send chat messages to each other without interrupting the lecture.

- All users can change their own expressions and their profile information.

The interaction between users in the public or private area will be different. In the public area, user can choose to talk to privately to one or publicly to all users that are presented in the room. Within private area, users will need to follow the rules that are set by the owner of the area, very much like the interactions within the virtual meeting room. The freedom of posting notes, changing one's avatar expression image or modifying one's personal setting or profile will be consistent across all areas.

User Interface

User interface should be easy to use, we propose that the personal expression can be changed with one mouse click or one key stroke. The major advantage of such a GUI is usability and simplicity, which can motivate interactions within a static environment.

A proposed user interface for the system is illustrated in Figure 6-1. The floating tool bar on the side is a way of accomplish user-friendliness, so any command can be issued with one mouse click. One can also map expressions to function keys on the keyboard to achieve a similar level of simplicity. In addition, When a user is talking to another in private, all other avatars presented in the room are shown in shadow. This is a helpful hint to make private conversation more apparent. The detailed user interface design is outlined in a later section. The figure shown here is only an initial assessment to the design.

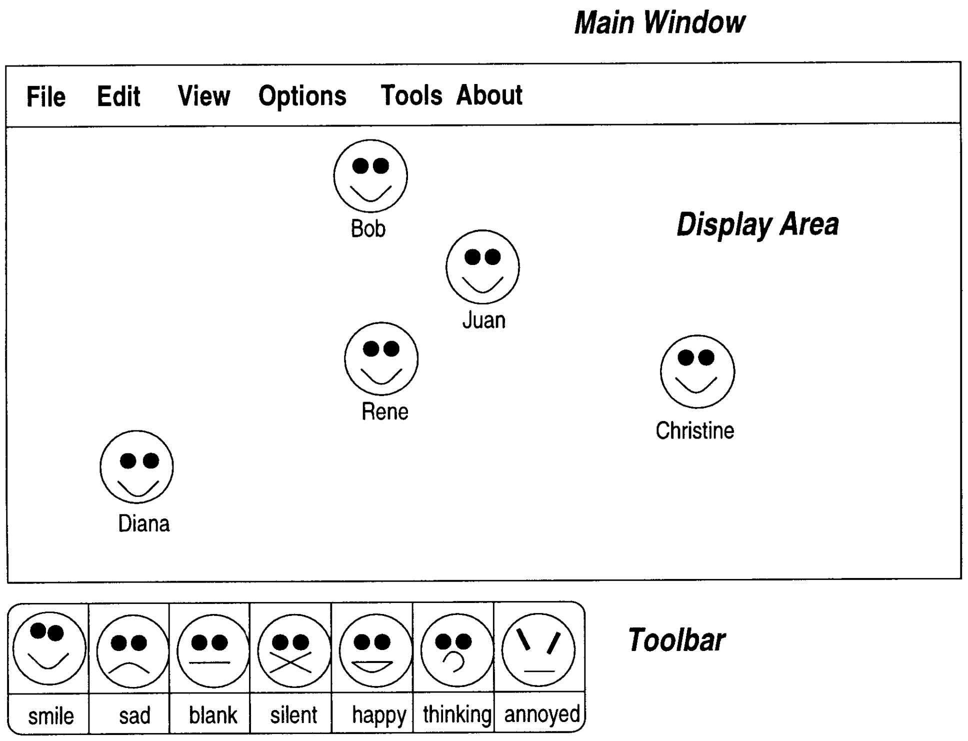


Figure 6-1: Suggested User Interface

Casual Interaction

The system should provide both object and event based casual interaction. There are spaces defined in which the user can establish an informal contact with the rest of users logged into the system. The Public Area as defined previously in this document provides an environment suitable for this kind of interaction. The different places (e.g. library, hallway, and coffee shop) may have distinct levels of privacy and availability. For instance, one person located at hallway could be more open to conversation engagement rather than a person at library. The avatars or cyberegos will play a important role in defining the availability of the user. So, a different set of cyberegos should be presented depending upon the location of the user in the virtual environment.

In this sense, typical casual interactions happens when user is not actively using the Cliq! application, but he is engaged in other network activities, such as email, ftp or telnet. Additionally, event and timely based casual interaction will be supported by keeping track of user's reactions. This would help to identify people's work habits, thus be a more useful tool for group dynamics. Proposed approaches for casual contact implementations are:

1. The system listens to TCP/IP ports (HTTP, SMTP, FTP, and TELNET) looking for activity related with services or applications associated to each port.

2. When user starts using an application, virtually move user to a given room where other people doing the same task is located.

3. When user is not reactive to the casual contact request, his appearance in the system will timed out. This option allow user to refuse to start a social interaction.

The main idea of this approach is to let users have knowledge of who is available. In most cases, users will only note the presence of another users but not actively interact with them. However every once in a while, the user may choose to initial a conversation with one another, as it happens when people run into each other at coffee machine or the water cooler in the physical world.

6.2 Feature List

The next milestone in the design phase is to come up with a detailed feature list from the scenario descriptions that was accomplished in the previous step. It is crucial to define the system's technical coverage and necessary goals, which gives clearer vision to the programmers as on what are the ones to achieve.

ENVIRONMENT

- **Main Hall:** This is the central location where everyone appears when they are available for conversation.
- Ways to enter:
  - Trigger action on PC any activity on TCP/IP ports or applications in use in the web browser (e.g. surfing, composer, news, and email)
  - Conscious effort by the user to enter the main hall.
- Ways to leave:
  - If entry was by trigger action, and user did not do anything for a certain period of time, between 1-10 minutes (user determines this length), then user is moved his own room.
  - If entry is deliberate entry, the user will stay until deliberate exit, until he/she logs off, or until he/she times out with inactivity.
- Possible actions:
  - Create a new room
  - Send a message to everyone in room
  - Send a message to one person in room
- Attributes:

  - Every room appears as a door in main hall. Depending on the type of room, the door will have a name, a list of people in room, a lock, and/or a form to search if a person is in room.
  - There is no limit to the number of rooms Conversations in the Main Hall will NOT be logged. Bulletin board can be used to post general announcements, such as message of the day.

- **Meeting Room:** This is a general purpose room which is visible from the main hall in which people can have meetings and private conversations.
- Attributes:
  - Whiteboard (which can be saved)
  - Name
  - Capacity
  - Access list of people allowed entry
  - Public knowledge of who is in the room? (yes/no)
  - Rules of conversation
  - All messages can be heard by everyone in room? (yes/no)
  - Room disappears when last person in room leaves. The text of all messages are saved, accessible only by people who participated.
- Default template rooms:
  - Private meeting: entry by invitation, no knowledge of whose in room, anyone can speak at any time, everyone in the room hears all messages
  - Chat room: anyone may enter, public knowledge of whose in room, anyone may speak at any time, private messages between two people allowed

  - Lecture hall: anyone may enter up to capacity, search to see if someone is in room (no visible list), only lecturer, or someone lecturer permits may speak.

- **Lecture Hall:** Special Meeting Room where there is a large audience and few presenters, e.g. a classroom.
- Attributes:
  - Only users that are on the meeting list (class list) can enter.
  - Conversation or messages can be logged.
  - Presenter (instructor) can lock the room and prohibit other users to exchange chat messages.
- **Private Room:** User's own work space where they can escape from other users.
- Attributes:
  - Every user has a private room.
  - Anyone that wants to enter other people's private rooms has to "knock", and can only enter those rooms that are not locked.
  - Users can not find out the list of users within the room if it is locked.

PROCESSES

- **Permanent Rooms:**
  - The main hall and private rooms of the users are permanently established.
  - Permanent rooms can not be destroyed by users.
  - If a user decides to cancel his registry in the system, the linked private room will be destroyed.

- **Creating a temporary room:**
  - Creating a room can only be done from main hall
  - Anyone may create a room
  - A user enters a room by clicking on the door and with the permission granted by the owner of the room.
  - The creator of a room chooses the format of the room. There are three templates as well as a custom option.
  - Room disappears when last person is left.
- **Casual contact:**
  - When a user triggers a network event, she appears in the Main Hall for some period of time. During this time, she will be present in the main hall and the client running on her computer will show all of the people present in the Main Hall.
  - After this period of time expires and the user has done nothing, the system ask to the user whether she prefers to be moved to his room or log out from the system.
  - The user may check to see who is in the main hall at any time, but this will cause her to appear in the main hall as well (it is a network event).
  - The user may start a conversation with anyone in the Main Hall by clicking on the person. Otherwise, any textual message is broadcasted to rest of users in the MainHall.
  - The user may also create a new room.
  - Once a user commits a manual action, they will remain in the cyberworld until they manually choose to leave, or until they timeout.
- **Intentional Contact:**

  - A user may choose to enter the main all at any time and stay there for as long as desired.
  - After 1 hour of inactivity by a user, the system will ask the user if they are present.
  - If no response is given, the user disappears from the cyberworld.

- **User Profile/Interaction**
  - When a user enters the system for the first time, a user profile is created, which specifies the causal interaction timeout, the persons name, and 5-10 pictures of the person to represent different facial expressions.
  - The facial expressions can be accessed at anytime by pressing a "hot key," the function keys on the keyboard for example.
  - The system will keep track of the last few messages sent by a user, these messages can be cycled through by using the up and down arrows, similar to how commands can be cycled through in most Unix shells.
  - Users will be able to playback conversations which they are a part of as well as the facial expressions or avoid this feature.

- **What this system will NOT do:**
  - This system will not handle audio or video, we already have applications to do that
  - This system will not allow document sharing other than the whiteboard in a room

6.3 Use Cases

Defining use cases is the third step toward a complete system design. It is a transitional phase between preliminary specification/feature list and detailed module design. By identifying *uses* at the beginning of the software development cycle, the entire process will benefit. *Uses* refer to the black box functionality of a program, only as what is seen from the outside at the user point of view. The internal structure of objects and of the system is not discussed in use cases. The purpose of use cases in this stage is to build a system model that is understandable by both the developers and the customers. A use case is a way to use the system. Users interact with a system by interacting with its use cases [18].

Use Cases follow two important rules [17]:

1. They capture a system's functional requirements.
   A use-case model defines a system's behavior through a set of use cases. The environment of the system is defined by describing the different users. The different users then operate the system through a series of use cases. Remember that the use-case model is an *external* view of a system, as opposed to the object model, which is an *internal* view of the same system.

2. They structure each object model into a manageable view.
   One view is drawn in each use case. A complete object model is seen through a set of object model views – one per use case (remember that objects are composed of data and functions which manipulate that data). In the most object-oriented methods, scenarios are used to find out if we had a complete definition of each object. Because use cases explore all possibilities of a *use*, we are guaranteed a complete object model by looking through all use cases in which the object has a role. In other words, every role of an object means a responsibility for the object. The total responsibility of an object is received by integrating all its responsibilities.

The use cases that were defined in the system are *User Registration*, *Modifying User Profile*, *User Login*, and *Create Room*, you will find the courses of action which illustrate the above use scenarios listed in Appendix A.

6.4 Module Separation and Task Division

Any design of a complex system needs some degree of modularity. It is in the best interests of the team to subdivide the system into different areas that are specific to a set of features. It came to our consensus that it was good to separate the social interaction and casual interaction into *user interface* module and *network* module respectively. How did such a decision make sense, one might ask. It is evident that both user interface and network transmission are crucial. Social interaction mainly is constrained in a limited space between parties, where casual interaction always have a bit of randomness involved, and the span of network increases such possibilities of achieving randomness. The following sections will present the design documents of the network and user interface modules.

6.5 Network Module Design

The proposed network model that handles the data traffic in the Cliq! system is a multi-client single server model. It can be expanded to a multi-server and multi-client architecture in a later version. To avoid unnecessary complexity, the choice made at the design level is to stay with a Client/Server model, and avoid clustering or meshing at the server side. The advantage of this model is its simplicity, it is a good implementation for an early prototype. The drawback of the model is its robustness, since the server will very much become the bottleneck when the number of supported clients increases.

6.5.1 System Architecture

The architecture of the Cliq! system will take advantage of the existing Internet infrastructure. Users of the system can use the service anywhere as long as s/he has access to the Internet. Figure 6-2 illustrates a broad view of the Clip! system architecture.

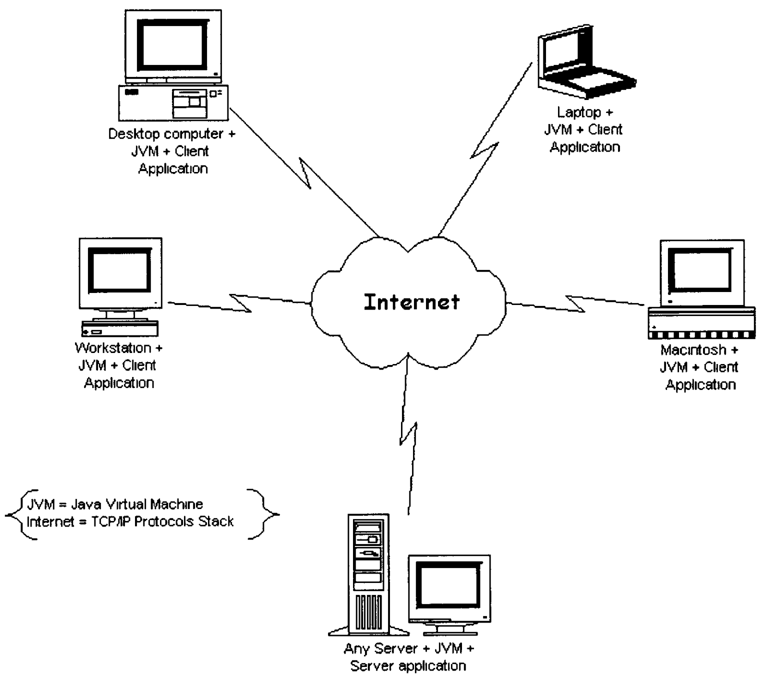

Desktop computer + JVM + Client Application
Laptop + JVM + Client Application
Internet
Workstation + JVM + Client Application
Macintosh + JVM + Client Application
JVM = Java Virtual Machine
Internet = TCP/IP Protocols Stack
Any Server + JVM + Server application

Figure 6-2: Cliq! System Architecture Overview

6.5.2 Network Design

To illustrate the proposed multi-client/single-server model, I will start with an example. In Figure 6-3, boxes that are on left are representing client machines, where as the server are located on right. The server side can be sub divide to two components, one is the *nameserver* which handles user registration, user login and logout. In addition, it also manages virtual environment (e.g. room) creation and destruction. The second component is the *netroom*, which handles the direct data exchange from client to client and acts as a broadcast intermediate. Consider *nameserver* as the main server, and netroom as the subserver.

In the Cliq! implementation, we will have the subserver module physically located on the main server. Later revisions can have those modules run as stand alone.

Assume we are logged in as client 1 (See Figure 6-3) and want to exchange messages with client 2 and client 3 who are sharing the same virtual user space. Client 1 will first log into the main server, the main server will keep a list of current users in the system, and a list of active rooms. The room object does not have to locate in the main server, it can be located in a subserver as long as it is registered with the main server. Similarly, Client 2 will log into the main server, same as Client 3. If they decide to go to the same room, the main server will provide the location of the subserver that contains the particular room object to all three clients. From this point on, the clients communicate with each other through the subserver.

In Figure 6-3, the gray line indicates a temporary connection, symbolizes for initial handshake. The solid line indicates a persistent connection. Since the clients only need to log in to the main server once, this action is done once, then the connection between the clients and the main server will terminate. The connections between clients and the subserver can last longer, depending on the duration of the client's stay.

6.5.3 Casual Interaction Implementation

The client side application should be running in a passive mode when user are not engaged in planned social interactions. Figure 6-4 illustrates the implementation model we had chosen for the casual interaction. The user of the system will have a permanent presence in the system, he stays in either two modes, one happens when he is actively engaged in social intentional interaction, the other one, passive or sleep mode occurs in other circumstances. There are pathways to move from passive to active mode automatically through user network activity generation. User will return to passive mode when he exists the main window.

The user interface of the passive mode is a small window with an action indicator and two buttons, shown in Figure 6-5. The passive client application will have an

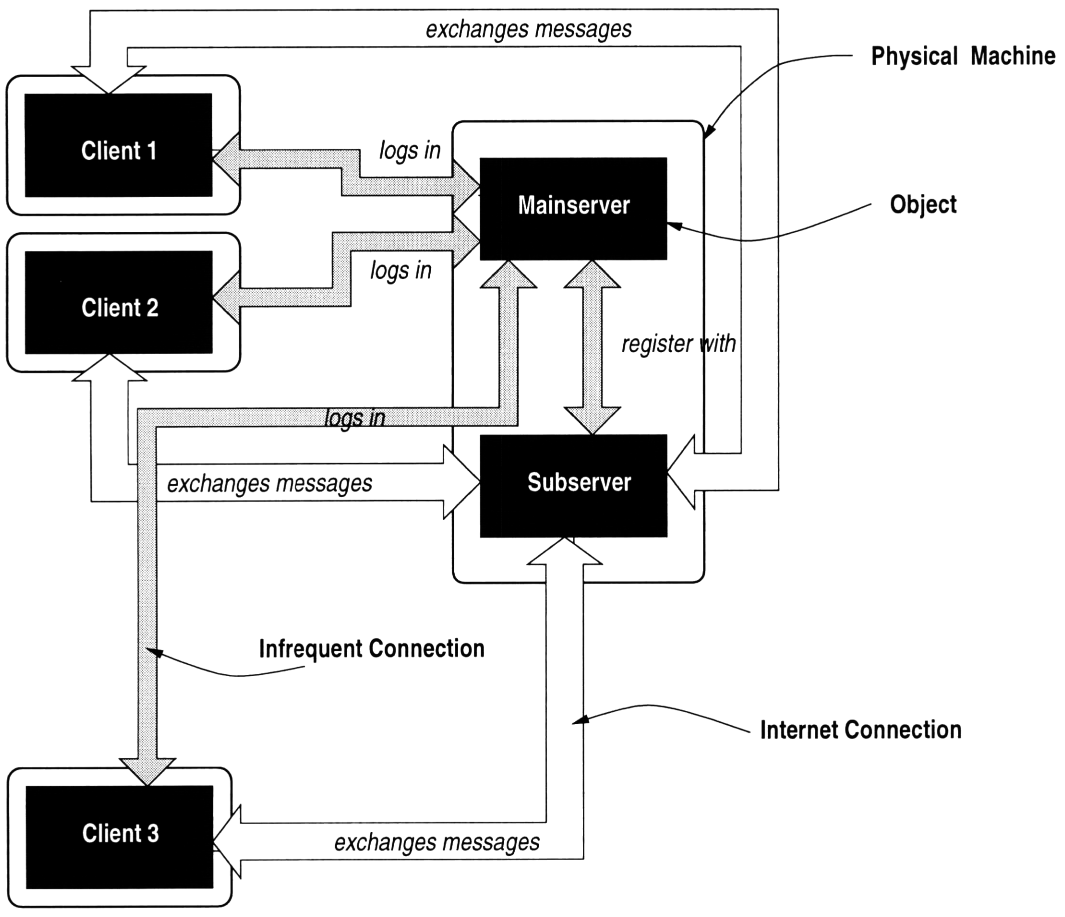


Figure 6-3: Client/Server Model of Cliq!

underlying network monitoring module that runs *netstat*[1] to obtain a snap shot of the client machine's network activities within small time intervals. The desirable range will be from 5 seconds to 30 seconds, the range will be decided by the programmer. This interval time can also be chosen by the users. Once the result of the netstat command is obtained, the client will check for certain events from the resulting data, such as web browsing, ftp events, fingering user or incorporating emails. Example of netstat output is shown in Table 6.1.

[1]netstat is a handy command that can be run on different operating systems to obtain a list of network port status.

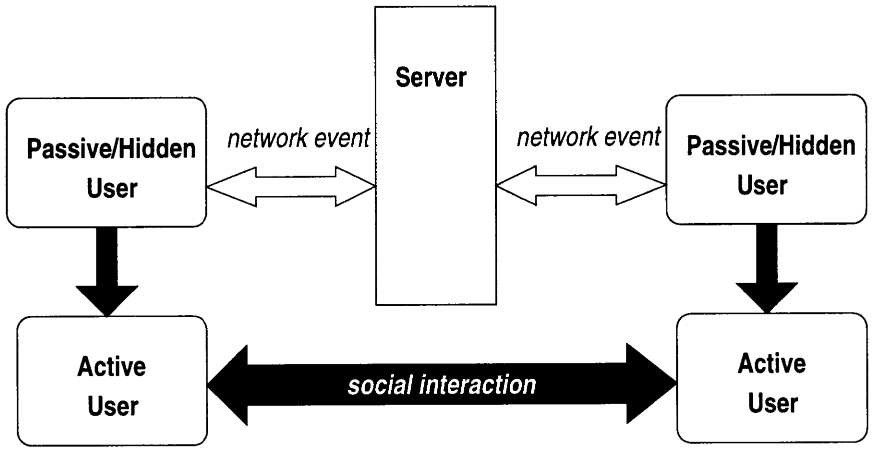


Figure 6-4: Casual Contact Interaction Model

Table 6.1: Netstat Output

Local Address	Remote Address	Swind	Send-Q	Rwind	Recv-Q	State
ml-142-1.MIT.EDU.40087	206.79.117.18.80	8760	0	8760	0	CLOSE_WAIT
ml-142.1.MIT.EDU.40088	209.1.234.200.80	8760	0	8760	0	ESTABLISHED
ml-142.1.MIT.EDU.40089	ad.doubleclick.net.80	8484	0	8760	0	ESTABLISHED
ml-142.1.MIT.EDU.40090	209.1.234.200.80	8760	0	8760	0	CLOSE_WAIT

Once such event is detected, the client will send a message to the connected server. The server application will keep a log file of all passive clients, and each network access notification from clients is queued and logged. If any notifications from different passive clients arrived within a fixed time interval, this threshold can be adjusted, the server will have the clients meet in the main hall (a permanent existence in the system). The user is responsible to activate the application, if s/he is not ready to participate in social contact, s/he can timed out from the casual engagement. If they choose to enter Cliq!, the status of the user will change from passive to active, followed by the social interaction user interface window popping up on the client machine.

Above is a simplified description of casual contact, the way that was implemented

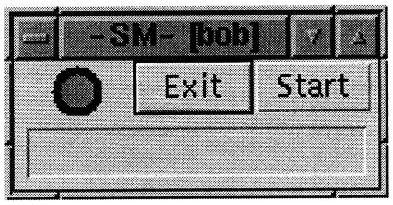


Figure 6-5: Casual Contact User Interface

in the system was a three stage process. The initial condition leads to casual contact is when the user is engaged in network activities, such as described above: telnet, ftp, web surfing, and emailing. The passive user will see a yellow light in the Cliq! casual contact user interface. The next condition happens, when there are other users in the system at the same time, the passive user's avatar icon will appear in the main hallway along with other active users. It would be a green light appear on the passive user's desktop along with a list of current active users. At this point, the passive user can decide to bring up the full bloom of the application or stay passive. The third condition comes forth when other users actively trying to talk to passive user, the passive will see a red light in s/he desktop to indicate the urgency, but s/he has the choice to continue what s/he is working by ignore the signal. The passive user will be timed out of the system if no actions are applied. Please see Figure 6-6 for details.

6.5.4 Object Diagram

The actual implementation of coordination of user interactions is broken down to two parts, as you are already familiar with the terms used above. These two parts are viewed as a solution to handle the two types of interactions specified in the analysis requirement: social and casual. To separate the two issue, one can consider a user is always presented in the system once he logs in to the main server, however he has the choice of being *active* or *passive*. A passive user need not to engage himself in any of the virtual interface environments. On the other hand, an active user will

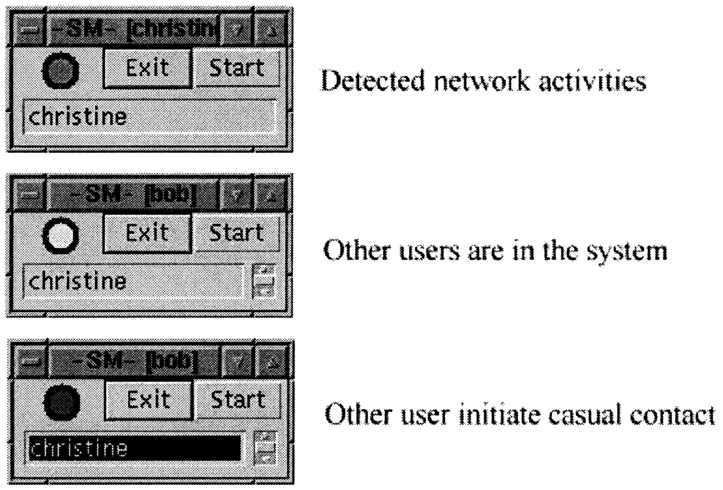

-SM- [christin
Exit
Start
christine
Detected network activities
-SM- [bob]
Exit
Start
christine
Other users are in the system
-SM- [bob]
Exit
Start
christine
Other user initiate casual contact

Figure 6-6: Three Conditions of Casual Contact

participate in the interactive meeting sessions. A passive user can become active if he uses network services, and by chance, these activities may invoke the casual interaction. Obviously, the passive user will also have the control to interact with others on his own, not just through some network-usage triggered events.

The server will keep a table of current active user list. The table should contain information of the client machine's IP address, the cyberego's current position on screen, and the room that the cyberego is shown. In addition, the server will keep a table of passive users, and the latest meaningful network event they generated. Most of the operations of moving users from room to room, broadcasting messages across rooms, or changing cyberegos on-screen position will involve updating and querying on these tables. Please see the data flow diagram provided in Figure 6-7.

The class Dgram and Serverdgram are both implemented based on the CAIRO system developed by Kareem Hussein[2]. These classes served as the lower level data

[2]The Collaborative Agent Interaction control and synchROnization (CARIO) System aims to bring together research on meeting and negotiation processes with distributed artificial intelligence concepts to explore methodologies for intelligent facilitation of distributed computer-supported meet-

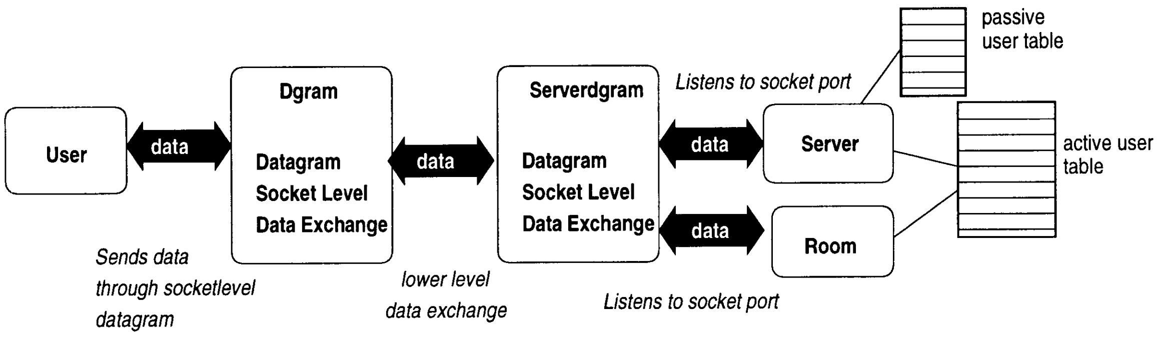

passive user table
Listens to socket port
Dgram
Serverdgram
User
data
Datagram Socket Level Data Exchange
data
Datagram Socket Level Data Exchange
data
Server
active user table
data
Room
Sends data through socketlevel datagram
lower level data exchange
Listens to socket port

Figure 6-7: Dataflow Diagram of the Client/Server Model

communication handler. We choose to reuse their declarations, since the underlying functionality of data transmission between agents are not different.

There are seven major classes shown in the network module object diagram (See Figure 6-8), they are *nameserver*, *netroom*, *netuser*, *serverRoot*, *dgram*, *serverdgram*, and *telement*. The data dictionary of the classes are provided in Appendix B.

6.6 User Interface Design

This document describes the overall organization of the system's user interface and social interaction, presents implementation issues, explains both strategic and tactical design decisions, and sets trade-off priorities.

6.6.1 User Expressions

The system will use an Avatar based for user representation within the environment. In social interaction, user expressions provide the mechanisms to know what is the person's emotional state. Emoticons (or "smileys") are sometimes a useful way of expressing emotional feeling in a text message. An emoticon is a symbol composed of a few text characters, and used as a kind of emotional shorthand to add meaning

ings. Please see http://ceel.mit.edu/ for further descriptions.

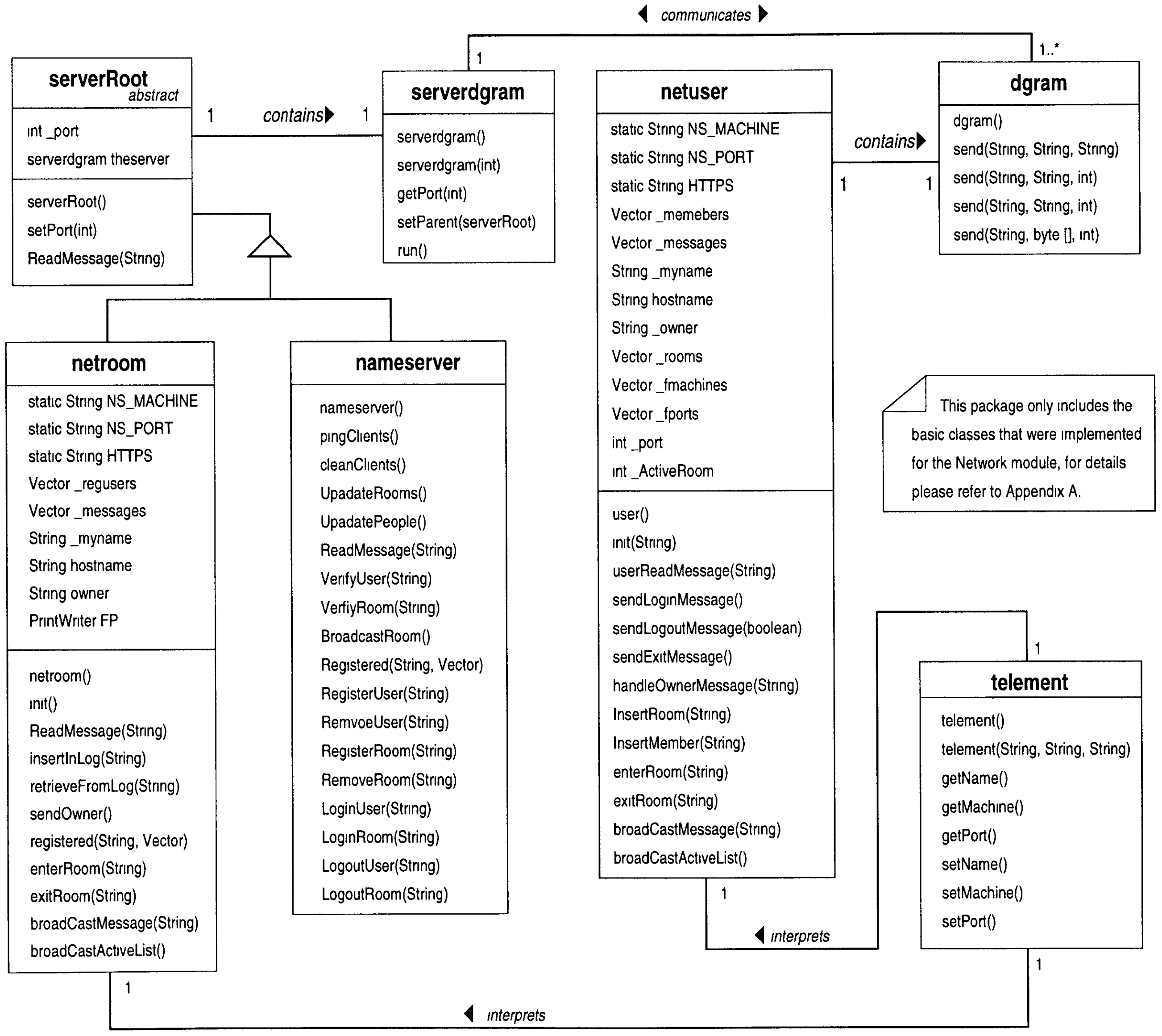


Figure 6-8: Network Module Object Diagram

to a message. For example, an emoticon may be used at the end of a comment to indicate that the comment was not intended to be taken seriously :-). Since most of the communication will be accomplished through textual messages, we propose an approach based in the identification of Emoticons symbols embedded in the message text to change user's Avatar expression. That is, we must parse text message posted by the user and if we find one, we should change the Avatar expression automatically according the symbol found. In addition, the system will provide a manual mode for change Avatar's expression using both keyboard shortcuts and a expression palette or toolbar (See Figure 6-9).

6.6.2 Event Management

As proposed in before, the system architecture is based in the Client/Server model. We believe that the client side should the responsible of (1) initiate peer to peer communication with the server, (2) translate user request into request for data from the server via a given protocol, (3) interact with user through the Graphical User Interface (GUI). In this sense, the client side must keep track of the users action/activities within the environment such the manipulation of objects displayed on screen, menus, toolbars or controls, and of those actions related with the interaction with other users (e.g. avatar movement, expression changing, and speaking). Additionally, the client should notify to the rest of the clients of the occurrence of all those events in order to maintain the consistency of the virtual environment throughout all the clients currently active. Besides that, the client must be able to handle the events that took place in the other clients and perform the actions associated with the events locally.

The mechanism for handling the events is described in Figure 6-10. User will interact with system and with other user through the client's GUI (a representation of the virtual environment) manipulating the different visual objects in the environment (e.g. rooms, objects in a room, and avatars), and with different widgets (e.g. controls, buttons, menus, and dialog boxes). All this object should be able to detect or listen

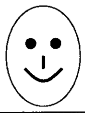
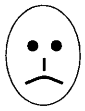
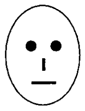
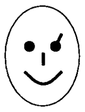
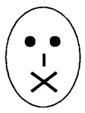
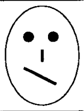
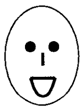
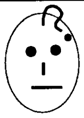
	**Smile** inflects user had made a pleasant statement
	**Sad** expresses concerns or sadness
	**Blank** indifferent or speachless
	**Wink** sarcastic remark or making a joke
	**Silent** lips are sealed, no comments
	**Annoyed** skeptical or disagreed
	**Big Smile** laughing,cracking up on a joke
	**Question** thinking or confused

Figure 6-9: Emotions and their Meanings in the System

different kind of events. For example, the avatars should be sensitive to mouse clicking and implement a method to report to the system the occurrence of this event, in such way that the system performs an adequate action. The Java language, in its version 1.1, defines a very useful event handling model. The 1.1 event handling model is based on the concept of an event listener. An objected interested in receiving events is an event listener. An object that generates events, an event source, maintains a list of listeners that are interested in being notified when a event occurs, and provides methods that allow listeners to add themselves and remove themselves from this list of interested objects. When the event source object generates an event, the event source notifies all the listeners objects that the event has occurred. An event source notifies an event listener object by invoking a method on it and passing it an event object. In order for a source to invoke a method on a listener, all listeners listener should implement the required method.

We propose to extend this model adding a little more functionality to the event listeners objects. The event listener must redirect to the server the events generated by the event source. In Figure 6-10, we have the description of how this mechanism will work. At the left side we have the user interaction with the different GUI objects, the event source objects (Rooms, avatars, and controls), this object must attach to their list of event listener an instance of the class EventManager, this class will implement most of the listeners interfaces defined in the Java API (e.g. ActionListener, MouseListener, and MouseMotionListener) and aggregates two additional methods: HandleEvent and SendEvent. When an event comes up, the object source passes it to the EventManager. The EventManager performs any action associated with the event if necessary and decides if this is a event that requires to be send to the rest of the clients or not. If it is a event that must be send to other clients (e.g. expression changing, avatar movement, and leaving the system), it creates the appropriated message and sends it to the server using the SendEvent method. The HandleEvent method will take all the incoming messages from the server and perform all the action necessary to fulfill the message instruction.

6.6.3 Protocol

The following is a description of the structure of the messages generated by the clients on the occurrence of a given event or request. The server should broadcast these events to the rest of the system clients connected to the server. The structure of the message is shown as in Table 6.2:

Table 6.2: Structure of User Interface Event Manager Message

4 Bytes	Variable Length
Event ID	Event Info

The first four bytes (the size of integer in Java) are the event identifier. Every event has a unique value associated in this field. The rest of the event message contains additional information related with each event. This part is variable both in content and length depending of the event. Most of messages includes includes either a user identifier (UID), a room identifier RID, or both. The UID is a unique number assigned to the each one of the user logged in the system. This number is assigned dynamically by the server when user logs in and he could could have a different number every time. This number is kept by the server in mapping tables storing the basic information about every user. This mapping table is detailed in the Table 6.3.

Table 6.3: Server Side Event Identification

UID	The user identification number	Integer
NICKNAME	The user nickname or username	String
IPADDRESS	The IP address from where the user is connected at	String
LOGINTIME	At what time the user logged in	String
LOCATION	This contains the RID where the user is located	Integer

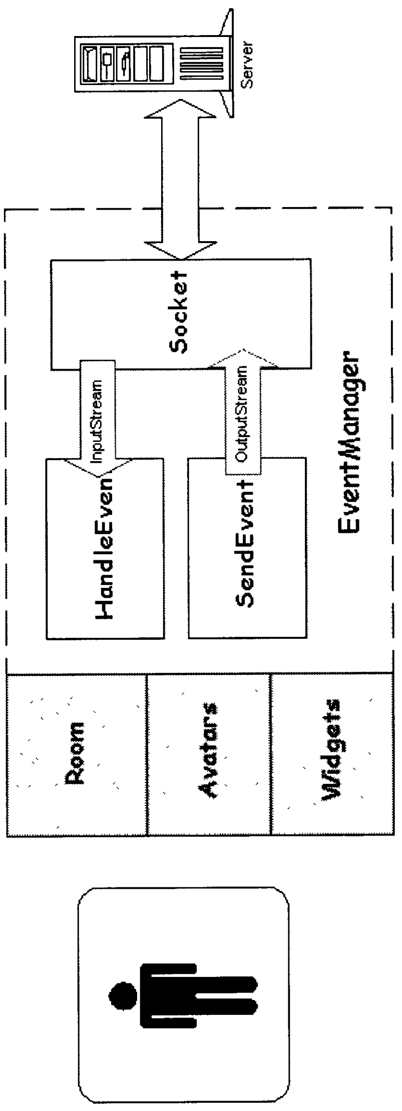


Figure 6-10: Relationships Between User Interface Classes

The room identifier is assigned as same as the user identifier is an unique number assigned dynamically to every room when it is created. This number is stored in mapping tables that holds the basic information about the rooms currently available. This mapping table is described in Table 6.4.

Table 6.4: Room Identifier Mapping

RID	The room identification number	Integer
CAPACITY	What is the capacity of the room	Integer
OWNER	The UID of the owner of the room	Integer
USERLIST	A list that holds the UID of all the users located in this room	Vector

6.6.4 Object Diagram

There are eight major classes shown in the user interface module object diagram (See Figure 6-11), they are *User*, *UIRoom*, *ImageLabel*, *Icon*, *Avatar*, *UserProfile*, *EventManager* and *EventMessage*. The data dictionary of the classes are provided in Appendix C.

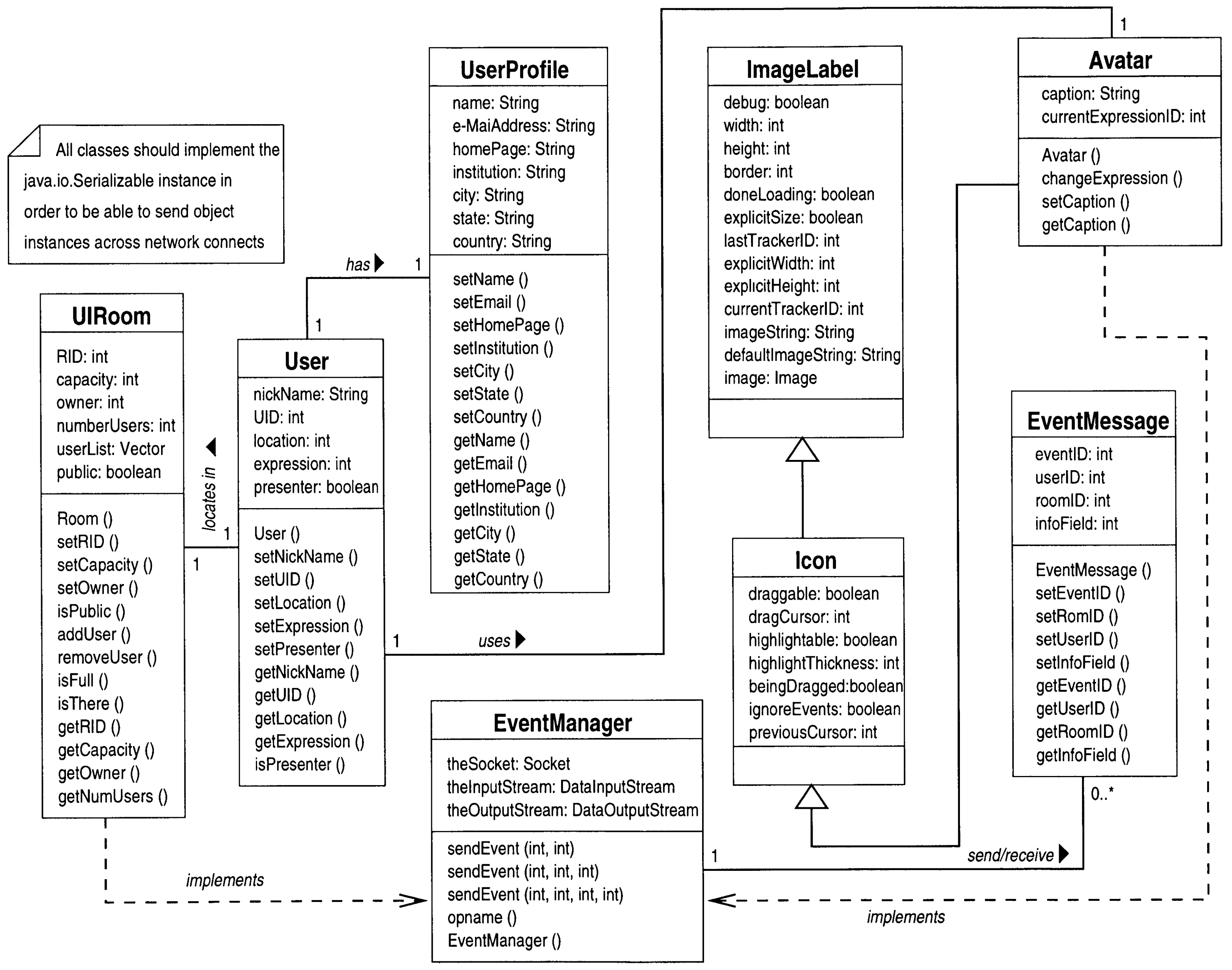


Figure 6-11: User Interface Object Diagram.

Chapter 7

Conclusion

In conclusion, the Cliq! system was implemented with success based on the design suggestion that were presented in this thesis. You shall find the user manual of Cliq! provided in Appendix D. Please don't hesitate to try out the system at your own convenience. It is important to have second opinion of what the future communication tool for software development should look like, since we all are going to create the future.

7.1 Project Technical Review

The design of the Cliq! system satisfied most of the features listed in the initial analysis requirement. However, there are numerous feature left out at the implementation because of some technical difficulties. The major accomplishment of the system is the development of casual interaction. Although in many areas the Cliq! was not very sophisticated in providing social interaction, the casual interaction feature was a great leap forward. It was the first attempt to bring a real world experience to the virtual environment.

As it was mentioned above, the unimplemented features are important, however, due to time constraints, they were left for next generations of the Cliq! system. The implementation of these features includes: replay of virtual environment activities, establishment of room ownership and access control, and differentiating user status (e.g. superuser). These features will improve the social interaction environment within the system. In addition, it is also important to focus on identifying features that are beneficial to enhance human contact. We do not wan to lose the purpose of this focused research when features are implemented for the sake of enhancement.

7.1.1 Future Improvement

The current Cliq! system does not support loading of multiple sets of user expressions. In the next version, dynamic image loading should be supported, so user can have the flexibilities to differentiate themselves from other users in the system. For example, the different set of emotions are applicable in different situations. In a staff meeting, the dress code is not formal, expressions are also less ambiguous. In contrast, an encounter at the hallway will allow more options in terms of what expressions can be used. Another technical feature that can be developed for the next generation Cliq! system is virtual image sharing, such that the user can add a new image to his expression set and immediately he can share it with others over the Internet.

Apart from purely technical issues illustrated above, as for future improvement, I would suggest the future DISEL class to use a different design model. The waterfall model was the software engineering process followed in the class. Such an model seems to be ineffective for the prototyping work that was conducted in the class. Students are locked in to a role that they chose in the beginning, and get less exposure to the overall project and software engineering process until much later. A suggested improvement of the class would be to expose different areas of software engineering process to all students in the beginning, have them understand the concepts through assignments. In addition, a student can be assigned to more than one role. The project will be divided to modules based on product features after the requirement analysis phase. For example, Member A could be a programmer of the network module and the testing engineer in the user interface module. Member B would be the designer of user interface module, and the quality control engineer of the network module. Because of the overlapping of responsibilities, the integration of modules at the end could be done a lot more smoothly. The illustration of this engineering process model described above is shown in Figure 7-1.

Requirement Analysis Phase
Identify the requirements of the system define functionalities, and separate the general concepts into small modules

Design/Testing/Quality Control
All team memebers are separated into module teams, each team composes of one designer, one programmer, one QC/V&V engineer and test engineer. Any member can play more than one role but these roles can not be the same role or within the same team. e.g. Designer1 can be Quality Control3.

Integration Testing/Debugging
All members will involve in this stage, desingers are responsible of coming up with integration design, and testing engineers are responsible of carrying out the executions.

Beta Testing/Modification
Beta testing is conducted through a larger user base, and new features can be added to different modules in this phase.

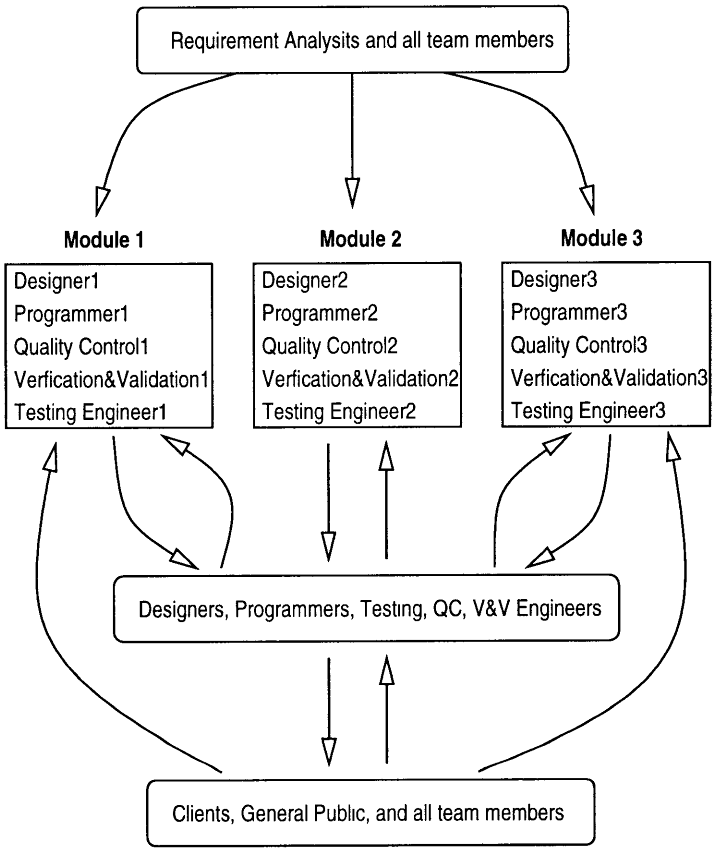


Figure 7-1: Suggested Software Engineering Model for Distributed Development

7.2 Collaborative Development Review

Since the team was working in a distributed environment, there were a lot of issues came up during the laboratory that was not entirely technically related. For those who are familiar with distributed team, they might had such experience as well. The gap could be caused by ineffective personal communication skills, culture differences, or other type of misunderstanding that was not immune in a co-locate environment. In a distributed environment, such issues remained maybe in a more sever form.

7.2.1 Understanding of Designer Role

I would like to start this section with my personal experience and illustrate the pros and cons of working in a distributed team. I would also back up my opinion with examples. First, let me present my understanding of the *Designer* Role in an interactive and cooperative team. As it is defined earlier in the discipline (see Chapter 3), designers play a crucial role in bringing and guiding the implementations of applicable solutions to real life problems. The designers are responsible to construct system that satisfies a broad range of issues, those include: functional specifications, limitations of target medium, performance and resource constraints, and business related restrictions, such as time and costs.

More importantly, the designers are not just working as individuals, but also as part of the constantly evolving team. The responsibilities of designers are spanned from creating system architecture to studying the feasibilities of any possible implementations. Therefore there are no fixed rules of how and what a designer should do to create the design. However there are guidelines and past experiences that will enhance the process. Generally, designers should use updated tools, correct assumptions, and complete requirements to start the designing cycle. Building on top of the analysts' solid requirements and other engineers' good intuitions, designers should be able to integrate these ideas together effectively.

7.2.2 Pros and Cons of Working in a Distributed Team

The distributed nature of this project brought in a few difficulties to all members. We weren't able to communicate with each other efficiently through non traditional methods, such as emailing, chatting or talking, versus making phone calls, face to face talking or video conferencing. As we all know, the advantages of having in-person communications are the instant feedback. There are always real time delay in exchanging emails or chatting on line. In addition, the instant feedback we get from talking to a person face to face are different from the feedback we get from reading an email. The information we can get from the facial expression or the body language of any person are tremendous, and most of these information could be lost if spoken words are the only messages that are transmitted.

One the positive side, the interactive nature of the team was actually beneficial to improve the design. As we were discussing the possibilities of bringing people from different technical backgrounds and communication skills together, we were talking about diversifying creativity and proficiency of the group. No one would doubt such diviersifications would help to achieve a better result. However such benefits of distributed team organization do not come without a price. The set backs of distributed work include: it is harder to coordinate schedules, to interact socially, and to communicate on the personal level.

Let me give you some examples to illustrate my points stated above. I have worked with Rene and Tim[1] on producing the Role Contract. I found it much easier to exchange ideas with Tim without writing them down on paper first. However, I needed to prepare notes every time I set up a talk session with Rene, because of our time constraints. I wanted to focus on talking about what we should discuss instead of running off to some digressions. In addition, the common geographic location made people feel closer, not just physically, but also mentally. Because Tim and I

[1]Tim was originally a member in DISEL's design team, he later on left for another research project

shared some common knowledge of the place that we live in, we could talk about the recent events on campus or engage in complaints of the MIT work loads. All these information were coming to us through different sources, such as the newspaper, words of mouth, or other media (TV, radio, and Internet). So without being aware of what is going in the cyberspace does not block anyone from knowing what has been happening in the world. Yet for distributed team, not being able to talk to each other through the computer network would almost terminate all means of communication.

Although my examples of pitfalls of a distributed team was discouraging, my out look to distributed project team is very bright. To work in an environment that computer communication is the dominate source, we can have a lot of flexibilities to interact. Besides of emailing, talking or chatting with the other party, teammates could have a common news source that was presenting a uniform information base to them automatically, think about the Push technology by PointCast[2] To reduce the complexity of the distributed environment, the team could be organized in a way that the contact persons are well defined. So instead of talking to the project manager for a requirement detail, designers can contact the analysts directly.

To further reduce the complexity of working in a distributed team, everyone should establish a relationship with everyone else. Through out the project, I was not limited to communicate only with my counterpart in CICESE, but also other ones that were working in the project. Through the group email list, I have heard opinions that are not directed related to me, but gave me insights on the possibilities of cyberego implementations. It helped me to think through my options as a designer. Such team dynamics and wide range of exposures to other ideas are not really visible in the common MIT classrooms. The web certainly make people feel freer to raise their voices on crucial topics, and these reactions are beneficial to our project (See Table

[2]Push technology is broadcasting based automatic information delivery with customized news categories. For example, users of Pointcast can customized their desktops, such that they will only receive the information that they are interested in. The news are automatically updated every 20 minutes, in the other words, the information is "pushed" to the user, user does not need to find it intentionally.

7.1 for summary).

Table 7.1: Pros and Cons of Working with a Distributed Team

Pros	Cons
Having more creative minds and innovative ideas that are not limited by nationality, geographical location, or cultures.	Opinions may not get across the table in real time.
Having diversified background in both education, working experience, and age.	Brainstorm or less structure sessions are harder to conduct.
Getting the best talents is more important than having people in one place.	Feedback may be misunderstood, because of the lack of body language and facial expression.
Indirect communication, such as email, are effective in a sense that people will be more concise and get the point across in the most effective way.	Indirect communication, such as email is asynchronous, and there is always a possibility that someone won't reply to the request.
Only the well organized materials are presented though the web and emails.	Easier to separate yourself from the people working at the other location and feel closer to the people working at the same site.
Internet talk can provide the interactive multi-channel communication.	Team members are more reserved about giving new ideas or raising controversial issues in fear of causing endless delays.

7.2.3 Communication Protocol

As I explored the possibilities of improving the distant communication, I came across some wonderful tools that could be easily access on the web: the discussing thread, the hypermail archive, and the new CAIRO application. Even if people chose to ignore emails, there are other good ways to obtain feedback. In addition, the team developed our own *communication protocols*, emails were separate in two categories with tags in the subject to indicate their importance. For example, tag [FYI] meant no immediate reply was needed, the email was purely for your information. On the other hand, tag [IMMEDIATE ACTION] meant urgent reply was mandatory, everyone should respond within 24 hours, it was used for high-prioritized decision making. We also developed our *presentation protocol*. The intermediate results of each phase (e.g. requirement analysis, design, programming, testing) were distributed to all members prior to its presentation in the weekly laboratory, and opinions originated from other group members were prepared before hand, so they could be presented at the same time. After the lab, these opinions/corrections were incorporated in the next presentation or document, such process iterated many times till the final documents are produced.

7.2.4 Summary

In a final note, I also found working in a cross-culture team challenging. The students in CICESE overcame barriers of language inconvenience to work with us gave me a lot of courage in seeing similar type of distributed team would work in the future. As many corporations went global, managing a diversified team was inevitable. I hope the experience I gained in this research will be beneficial to both the academic and commercial environment.

7.3 Final Thought

As a departing thought, I would like to bring everyone back to the core of this research effort: collaborative learning over distance. Through DISEL, we had achieved an unique prototype of distance communication tool. It helped to improve geographically distant parties to better realize common goals, streamline workflow, and most importantly to form lasting bonds. Nevertheless, collaborative learning research requires such continue effort to fuel its next breakthrough.

Appendix A

Use Case Definitions

There are four use cases describe here, each is an extension of the scenario descriptions described in Chapter 6. In addition, each use case contains three tables. The first table is a brief description of the actor[1] and primary actions conducted by the actor. The second table shows the possible outcomes of those actions. The last table depicts the alternative outcomes and other dynamics between the actor and the system.

Table A.1: User Registration Use Case

Primary Actors:	User	**Secondary Actors:**	N/A
Assumptions:	User can be identified as a person in general. The meeting participant, presenter/assistant is entirely environment dependent, since there is no need to have special privileges for anyone in a casual environment.		
Scope & Level	Primary task		
Success End Condition	An user profile or identity is created in the system.		
Failed End Condition	User can't log into the system.		
Limitations:	N/A		
Users:	N/A	**Extends:**	N/A

[1]An actor represents the user of the system.

Table A.2: Ideal Course of Action for User Registration

Step	**Action**
A	Actor runs client application to access the system's user registration service. Client displays an information capture window.
B	Actor supplies information requested in the window. 1. Client request a connection to user registration server. 2. Server accepts connection. 3. Client submits information. 4. Server accepts, verifies, and validates information. 5. Server creates the user profile and sends an acknowledge. 6. Client displays successful fulfillment message.
C	Actor chooses to exit the process represented by this use case. System exits this use case, removing all remaining related displays.

Table A.3: Alternate Courses of Action for User Registration

Course	**Step**	**Action**
1	B	CONDITION: *Client can't establish a connection with the server.* Actor supplies information requested at display. 1. Client request a connection to user registration server. 2. Server refuse connection or client gets no responses from server. 3. Client notifies actor that service is not available.
2	B	CONDITION: *Inconsistencies in information submitted by actor* Actor supplies information requested at display. 1. Client request a connection to user registration server. 2. Server accepts connection. 3. Client submits information. 4. Server accepts, verifies and validates information and finds inconsistencies or errors in the information submitted by the actor. 5. Server sends back a negative acknowledgment to the client. 6. Client notifies the error to actor and prompts for correction.

Table A.4: Modify User Profile Use Case

Primary Actors:	User	**Secondary Actors:**	N/A
Assumptions:	User can be identified as a person in general. The meeting participant, presenter/assistant is entirely environment dependent, since there is no need to have special privileges for anyone in a casual environment.		
Scope & Level **Success End Condition**	Primary task Information stored in the user profile or identity is updated.		
Failed End Condition	Information must remain unchanged.		
Limitations:	N/A		
Uses:	N/A	**Extends:**	N/A

Table A.5: Ideal Course of Action for Modifying User Profile

Step	Action
A	Actor runs client application to access the system's user profile update service. Client request actor's username and password.
B	Actor supplies a user name and password. 1. Client request a connection to user registration server. 2. Server accepts connection. 3. Client submits user name and password and request the actor's profile information. 4. Server authenticates user identity, authentication succeeds and sends user profile information to the client. 5. Client receives information from the server, close the connection, and displays it to be modified by the actor.
C	Actor edits information. 1. Client request a connection to user registration server. 2. Server accepts connection. 3. Client submits information. 4. Server accepts, verifies, and validates information. 5. Server updates the user profile, sends an acknowledgment to the client, and closes connection. 6.Client displays successful fulfillment message
D	Actor chooses to exit processing represented by this use case. System exits this use case, removing all remaining related display(s).

Table A.6: Alternate Course of Action for Modifying User Profile

Course	Step	Action
1	B	CONDITION:*Client can't establish a connection with the server.* Actor supplies information requested in the window. 1. Client request a connection to user registration server. 2. Server refuse connection or client gets not response from server. 3. Client notifies actor that service is not available.
2	B	CONDITION: *Actor's identity authentication fails.* Actor supplies a user name and password. 1. Client request a connection to user registration server. 2. Server accepts connection. 3. Client submits user name and password and request the actor's profile information. 4. Server authenticates user identity,authentication fails, and sends a negative acknowledgment to the client. 5. Client receives the negative acknowledgment, notifies to the actor, and request a user name and password again. Use case continues with *Ideal Course of Action*, step B.
3	C	CONDITION: *Client can't establish a connection with the server.* Actor supplies information requested in the window. 1. Client request a connection to user registration server. 2. Server refuse connection or client gets not response from server. 3. Client notifies actor that service is not available.
4	C	CONDITION:*Inconsistencies in modifications submitted by actor.* Actor supplies information requested in the window. 1. Client request a connection to user registration server. 2. Server accepts connection. 3. Client submits information. 4. Server accepts, verifies and validates information, finds inconsistencies or errors in the information submitted by the actor. 5. Server sends back a negative acknowledgment to the client and closes connection. 6. Client notifies the error to actor and prompts for information correction. Use case continues with *Ideal Course of Action*, step C.

Table A.7: User Login Use Case

Primary Actors:	User	**Secondary Actors:**	N/A
Assumptions:	The user is registered in the system before log in into the system. Initially, when the user logged in the avatar will appear in the main hallway. The user can request to enter to a meeting room. If the users logs in as presenter and tries to enter to a meeting room, the system must check whether the presenter has the ownership for this meeting room. If the user logs in as meeting participant the system must check whether the user is not registered for the session and if the area is locked or overfilled. User can be identified as a person in general.		
Scope & Level	Primary task		
Success End Condition	User can access the virtual environment.		
Failed End Condition	User can't access the virtual environment.		
Limitations:	N/A		
Users:	N/A	**Extends:**	N/A

Table A.8: Ideal Course of Action for User Login

Step	Action
A	Actor runs client application to access the virtual environment. Client requests an user name and password to verify user identity.
B	Actor supplies an user name and password. 1. Client request a connection to user access server. 2. Server accepts connection. 3. Client submits the user name and password and request the user's profile information and the information about current state of the virtual environment. 4. Server authenticates user identity, authentication succeeds. 5.Server sends the user profile, information about current state of the virtual environment to the client and closes connection. 6. Client initializes virtual environment. 7. Client requests a connection to start sending/receiving events to the server. 8. Client starts keep tracking of user generated events.
C	Actor chooses to leave the system. System exits this use case, closing connection with the server and removing all remaining related display(s).

Table A.9: Alternate Course of Action for User Login

Course	Step	Action
1	B	CONDITION:*Client can't establish a connection with the server.* Actor supplies an user name and password. 1. Client request a connection to user access server, request the user's profile information and the current state of the virtual environment. 2. Server refuse connection or client gets not response from server. 3. Client notifies actor that service is not available.
2	B	CONDITION:*Actor's identity authentication fails.* Actor supplies an user name and password. 1. Client request a connection to user access server, request the user's profile information and the current state of the virtual environment. 2. Server accepts connection. 3. Client submits the user name and password and request the user's profile information and the information about current state of the virtual environment. 4. Server authenticates user identity, authentication fails, and sends a negative acknowledgment to the client. 5. Client receives the negative acknowledgment, notifies to the actor, and request a user name and password again.

Table A.10: Create Room Use Case

Primary Actors:	User	**Secondary Actors:**	N/A
Assumptions:	Any user may create a room. Rooms are persistent, and only the user who created the room may destroy it. User must be in the main hall to create a new room. The creator of the room chooses the set up of the room. There are three templates rooms as well as a custom option. The information about the rooms is kept in the user profile. There is no limit in the number of rooms.		
Scope & Level	Primary task		
Success End Condition	A new room is created within the virtual environment.		
Failed End Condition			
Limitations:	N/A		
Users:	N/A	**Extends:**	N/A

Table A.11: Ideal Course of Action for Create Room

Step	**Action**
A	Actor chooses to create a new room. Client application request information about the rooms configuration.
B	Actor supplies the room configuration to the client. 1. Client submits new room configuration to the server. 2. Server validates and verifies room configuration. If the room configuration is accurate, saves the configuration in the user profile and notify to all clients the creation of a new room. 3. Client receives notification from server and creates the room.

Table A.12: Alternate Course of Action for Create Room

Course	**Step**	**Action**
1	B	CONDITION:*Rooms configuration is inaccurate or invalid* Actor supplies the room configuration to the client. 1. Client submits new room configuration to the server. 2. Server validates and verifies room configuration. If the room configuration is invalid, sends a notification to client. 3. Client receives notification from server and request the information again to the actor.

Appendix B

Network Module Data Dictionary

Class: nameserver

This class keeps track of all network objects in the system.
Extended from serverRoot

Public Methods:

nameserver () :
Constructor method, starts nameserver.
pintClients () :
Pings users to make sure no have died unexpectedly.
cleanClients () :
Removes dead clients (users)
UpdateRooms () :
Displays list of current rooms.
UpdatePeople () :
Displays list of current users.
ReadMessage (String) :
Process the message received by the serverRoot.
VerifyUser (String) :
Handles ping replies from users.
VerifyRoom (String) :
Handles ping replies from rooms.
BroadcastRooms () :
Sends the list of rooms to all users.
Registered (String, Vector) :
Checks to see if the string is a element of the vector (helper method)
RegisterUser (String) :
Adds user to the nameserver user list. User gets a Unique ID.
RemoveUser (String) :
Removes user from the nameserver user list.
RegisterRoom (String) :
Adds room to the nameserver room list.
RegisterRoom (String) :
Adds room to the nameserver room list.
RegisterRoom (String) :
Adds room to the nameserver room list.
RemoveRoom (String) :
Removes room from the nameserver room list.
LoginUser (String) :
Notes that a user is entering a room.
LoginRoom (String) :
Notes that some is in the room.

Class: netroom

The basis for a network "room" object. Class must be extended to include a user interface.
Extended from serverRoot

Private Variables:

NS_MACHINE : static String
Nameserver machine IP Address
NS_PORT : static String
Nameserver port
HTTPS : static String
URL server string (In case we need to load images or something)
_regusers : Vector
Users in the room
_messages : Vector
Messages to be processed
_myname : String
Name of the room
hostname : String
Machine name (IP address) on which the room resides
owner : String
Name of the creator of the room
FP : PrintWriter
Pointer to log file

Public Methods:

netroom () :
Constructor method.
init () :
Initializes the network communications. Parameter order: title, creator.
roomReadMessage (String) :
Handles basic communication messages. When room is extended, this method should be envoked by ReadMessage.
insertInLog (String) :
Logs the string int the log file. Should only be called through messages.
retrieveFromLog (String) :
Sends all log entried to users in the room. Should only be called through messages.
sendOwner () :
sends the owner (creator) of the room to current users in the room.
regiestered (String, Vector) :
Checks to see if the string is a element of the vector (helper method).
enterRoom (String) :
Adds user to the room. Should only be called through messages.
exitRoom (String) :
Removes user from the room. Should only be called through messages.
broadCastMessage (String) :
Sends the string to all users.
broadCastActiveList () :
Send to all users (in the room) a list of current users (in the room).

Class: netuser

The basis for a network "user" object. Class must be extended to include a user interface.

Private Variables:

NS_MACHINE : static String
Nameserver machine IP Address
NS_PORT : static String
Nameserver port
HTTPS : static String
URL server string (In case we need to load images or something)
_members : Vector
Users in the room with you
_messages : Vector
Messages to be processed
_myname : String
Name of the user
hostname : String
Machine name (IP address) on which the room resides
_owner : String
Name of the creator of the room you are in
_rooms : Vector
Current rooms that available
_fmachines : Vector
Machines on which current rooms reside
_fports : Vector
Ports on machines on which current rooms reside
_port : int
User's port number
_ActiveRoom : int
Current vector index that coresponds to your current room

Public Methods:

netuser () :
Constructor method.
init (name : String) :
Initializes the network communications.
userReadMessage (String) :
Handles basic communication messages. When user is extend,
this method should implement serverRoot.ReadMessage (String).
sendLoginMessage () :
Sends request to login to nameserver.
sendEnterRoomMessage () :
Sends Enter Room to the room specified by _ActiveRoom.
sendLogoutMessage (boolean) :
Sends messages to the room specified by _ActiveRoom and the nameserver
notifing them that the user is leaving. The boolean parameter should be
true. (false only if you are kicked out by the room itself).
sendExitMessage () :
Sends messages to the room specified by _ActiveRoom and the nameserver
notifing them that the user is exiting the system.
insertRoom (String) :
Adds a room to the list of rooms. Should only be called through messages.
insertMember (String) :
Adds user to the current room. Should only be called through messages.
enterRoom (String) :
Adds user to the room.
exitRoom (String) :
Removes user from the room.
broadCastMessage (String) :
Sends the String to all users.

Class: serverdgram

A class that is used by a serverRoot to listen to the active port.
It envokes serverRoot.ReadMessage(String) when DatagramPackets arrive.
Extended from Thread

Public Methods:

serverdgram () :
Constructor method.
serverdgram () :
Constructor method with specified port value.
getPort (int) :
Returns the port value or the parent serverRoot.
setParent (serverRoot) :
All DtatgramPackets are sent to the specified serverRoot.
run () :
Actual listening to the port, envoked with Thread.start().

Class: serverRoot

A base class for actual network objects (e.g. rooms and users) with UI.
Extended from Frame

Private Variables:

_port : int
port value for the serverRoot
theserver : serverdgram
The serverdgram that does the port listening

Public Methods:

serverRoot () :
Constructor method.
setPort (int) :
Sets the port value for the serverRoot.
ReadMessage (String) :
Process the message received by the serverRoot.

Class: telement

A base class used to represent network objects (e.g. rooms and users).
It can be extended to include other functionality.

Public Methods:

telement () :
Constructor method.
serverdgram (String, String, String) :
Constructor method with specified name, port, and machine value.
getName () :
Returns the name the the telement.
getPort () :
Returns the port of the telement.
getMachine () :
Returns the machine of the telement.
setName () :
Sets the name the the telement.
setPort () :
Sets the port of the telement.
setMachine () :
Sets the machine of the telement.

Class: dgram

This class sends DatagramPakcets over specified port.

Public Methods:

dgram () :
Constructor method with no port value.
dgram(int) :
Constructor method with specific port.
send (message : String, machine : String, port : int) :
Sends the message.
send (message : String, machine : byte [] , machine : int) :
Sends the message.

Appendix C

User Interface Module Data Dictionary

Class: User

This class is used to hold the basic information about the state of a given used within the environment.

Private Properties:

nickName : String
The user name or user's nickname.
UID : int
User's identification number
location : int
The RID of the room where the user is located.
expression : int
Holds the code for the user's current expression.
presenter : boolean
Indicates whether the user is a presenter or a normal audience. Its value is true when the user is presenter.

Public Methods:

User () :
Constructor method.
setNickName (theNick : String) :
Sets the value of the attribute nickName.
setUID (theUID : int) :
Sets the value of the attribute UID.
setLocation (theRID : int) :
Sets the value of the attribute location.
setExpression (theExp : int) :
Sets the value of the attribute expression.
setPresenter (theState : boolean) :
Sets the value of the attribute presenter.
getNickName () : String
Returns the value of the attribute nickName.
getUID () : int
Returns the value of the attribute UID.
getLocation () : int
Returns the value of the attribute location.
getExpression () : int
Returns the value of the attribute expression.
isPresenter () : boolean
Returns the value of the attribute presenter.

Class: UserProfile

This class holds the information about all users registered in the system.

Private Properties:

name : String
User's full name
e-MailAddress : String
User's e-mail address
homePage : String
URL pointing to user's personal homepage
institution : String
The institution where the user belongs to (e.g. Company name, university, college,etc.).
city : String
The name of city where the user is located.
state : String
The name of the state where the user is located
country : String
The name of the country where the user is located.

Public Methods:

setName (name : String = default) :
Sets value of the attribute name.
setEmail (email : String = default) : return
Sets the value of the attribute e-MailAddress.
setHomePage (thePage : String = default) : return
Sets the values of the attribute homePage
setInstitution (theInstitution : String = default) : return
Sets the values of the attribute institution
setCity () :
Sets the values of the attribute city.
setState (theState : String) :
Sets the values of the attribute state
setCountry (theCountry : String) :
Sets the values of the attribute country.
getName () : String
Returns the value of the attribute name.
getEmail () : String
Returns the value of the attribute e-MailAddress.
getHomePage () : String
Returns the value of the attribute homePage
getInstitution () : String
Returns the value of the attribute institution.
getCity () : String
Returns the value of the attribute city.
getState () : String
Returns the value of the attribute state.
getCountry () : String
Returns the value of the attribute country.
UserProfile () :
Constructor method.

Class: Avatar

This class is the visual representacion of the user within the environment.
It's the responsible for the detection of events generated by the user.
Derived from Icon

Private Properties:

caption : String
The caption to be displayed with the avatar.
currentExpresionID : int
The code of the current expresssion.

Public Methods:

Avatar () :
Constructor method.
changeExpression (expresionID : int) :
Changes the expression of the avatar.
setCaption (theCaption : String) :
Sets the value of the attribute caption.
getCaption () : String
Returns the value of the attribute caption.

Class: EventManager

This class implements the event handling mechanism.

Private Properties:

theSocket : Socket
theInputStream : DataImputStream
theOutputStream : DataOuptStream

Public Methods:

sendEvent (eventID : int, param1 : int) : return
sendEvent (eventID : int, param1 : int, param2 : int) :
sendEvent (eventID : int, param1 : int, param2 : int, param3 : int) :
opname (argname : argtype = default) : retur
EventManager () :

Class: Icon

Derived from ImageLabel

Public Properties:

draggable : boolean
dragCursor : int
highlightable : boolean
highlightThickness : int

Private Properties:

beingDragged : boolean
ignoreEvents : boolean
previousCursor : int

Public Methods:

Icon () : void
Icon (: java.net.URL) : void
Icon (: java.awt.Image) : void
mouseUp (: java.awt.Event, : int, : int) : boolean
Icon (: String) : void
paint (: java.awt.Graphics) : void
mouseDrag (: java.awt.Event, : int, : int) : boolean
mouseDown (: java.awt.Event, : int, : int) : boolean
handleEvent (: java.awt.Event) : boolean
handleIconEvent (: java.awt.Event, : java.awt.Container) : boolean
componentUnder (: int, : int, : java.awt.Container) : java.awt.Component

Private Methods:

report (: String, : int, : int) : void
parentFrame () : java.awt.Frame
iconBeingDragged (: java.awt.Container) : Icon

Class: Room

Private Properties:

RID : int
The room identification number.
capacity : int
The maximum capacity of the room.
owner : int
Contains the UID of room's owner.
numberUsers : int
The number of users currently located in the room.
userList : Vector
A vector tha holds a list of User class instances of the user currently located in the room.
public : boolean
This attribute indicates whether the room is public or not.

Public Methods:

Room () :
Constructor method
setRID (theRID : int) :
Sets the value of the attribute RID.
setCapacity (theCapacity : int) :
Sets the value of the attribute capacity.
setOwner (theOwner : int) :
Sets the value of the attribute owner.
isPublic () : boolean
Tells whether the room is public or not. Returns true when the room is public.
addUser (theUID : int) :
Adds a new user to the room (A new user enters the room).
removeUser (theUID : int) :
Remove a user from the room (the user leaves the room).
isFull () : boolean
Indicates whether the room has reached its maximum capacity.
Returns true if room has reached its maximum capacity.
isThere (theUID :) : boolean
Tells if a given user is in the room. Returns true if the user is in the room.
getRID () : int
Returns the value of the attribute RID.
getCapacity () : int
Returns the value of the attribute capacity.
getOwner () : int
Returns the value of the attribute owner.
getNumUsers () : int
Returns the value of the attribute numberUsers.

Class: ImageLabel

Derived from Canvas

Private Properties:

debug : boolean
width : int
height : int
border : int
doneLoading : boolean
explicitSize : boolean
lastTrackerID : int
explicitWidth : int
explicitHeight : int
currentTrackerID : int
imageString : String
defaultImageString : String
image : Image

Public Methods:

ImageLabel () : void
inside (: int, : int) : boolean
resize (: int, : int) : void
reshape (: int, : int, : int, : int) : void
centerAt (: int, : int) : void
getWidth () : int
getHeight () : int
getBorder () : int
setBorder (: int) : void
isDebugging () : boolean
waitForImage (: boolean) : void
setIsDebugging (: boolean) : void
ImageLabel (: java.net.URL) : void
ImageLabel (: java.awt.Image) : void
ImageLabel (: String) : void
getImage () : java.awt.Image
paint (: java.awt.Graphics) : void
getBorderColor () : java.awt.Color
setBorderColor (: java.awt.Color) : void
minimumSize () : java.awt.Dimension
preferredSize () : java.awt.Dimension
ImageLabel (: java.net.URL, : String) : void
getDefaultImageString () : String
setDefaultImageString (: String) : void
Protected Methods:
hasExplicitSize () : boolean
debug (: String) : void
getImageString () : String
drawRect (: java.awt.Graphics, : int, : int, : int, : int, : int, : java.awt.Color) : void

Private Methods:

¡clinit¿ () : void
makeURL (: String) : java.net.URL
loadImage (: java.net.URL) : java.awt.Image
makeURL (: java.net.URL, : String) : java.net.URL

Class: EventMessage

This class encapsulates the information in the event message sent to server
and event messages received from the server.

Private Properties:

eventID : int
The event identifier. It holds the code to identify what kind of message it is.
userID : int
This attribute holds the user ID number of a given user name. If the event message doesn't include the user ID its value is zero.
roomID : int
This attribute holds the room ID number of a given user room. If the event message doesn't include the room ID its value is zero.
infoField : int
This field holds additional information in some event messages. If the event message doesn't requieres additional information its value is zero.

Public Methods:

EventMessage (theEventID : int) :
Constructor method
EventMessage (theEventID : int, theUserID : int) :
Constructor method
EventMessage (theEventID : int, theUserID : int, theRoomID : int) :
Constructor method
EventMessage (theEventID : int, theUserID : int, theRoomID : int, theInfoField : int) :
Constructor method
setEventID (theEventID : int) :
Sets the value of the attribute eventID
setRoomID (theRoomID : int) :
Sets the value of the attribute roomID.
setUserID (theUserID : int) :
Sets the value of the attribute userID.
setInfoField (theInfoField : int) :
Sets the value of the attribute infoField
getEventID () : int
Returns the value of the attribute eventID.
getUserID () : int
Returns the value of the attribute userID.
getRoomID () : int
Returns the value of the attribute roomID.
getInfoField () : int
Returns the value of the attribute infoField

Appendix D

User Manual

D.1 Installation

The installation of the system is divided in two parts: the installation of the server and the installation of the client or application.

D.1.1 Server Installation

To install the server, copy the classes into a directory. The server uses the version 1.1.5 of jdk so ensure that this version is available in your computer.
Modify the file "autoexec.bat" to include the path to the java libraries and executables, and the classpath to the java classes. For example:

path
drive[1] :/*java_dir*[2]/lib;drive:/*java_dir*/bin

classpath
drive:/*java_dir*/bin/classes.zip

If these lines do not appear, include them. Edit file *room.config* to modify the name of the machine where the server will run. Ensure that the server_machine name used is correct. Don't modify the port number. For example:

server_machine = *servername*[3]

When the instructions are completed, the server is ready to run.

[1]drive is the drive where the jdk ver. 1.1.5 is.

[2]java_dir is the name of the directory where jdk ver. 1.1.5 is.

[3]servername is the name of the machine that the server application is going to run on, e.g. cee1.mit.edu

D.1.2 Client Installation

To install the client, copy the files into a directory. Depends on the machine in which the client will be executed run the following file:

Operating System	**Installation File**	**Usage**
Win95	install.bat	*install*
SUN	install.sun	*source install.sun*
SGI	install.sgi	*source install.sgi*

The execution will recompile a file and create a directory for the images. Once the process is finished the user is ready to run the client application.

D.2 User Registration

The system administrator is in charge of registering users, he/she will provide a username and password that the user can use to enter the system. To be registered please contact your system administrator. If you are interested in setting up Cliq! system on your own, create a password file *passwd*, you can find a sample in the code package.

D.3 Server

The server application is in charge of managing "users" in the system. This program keeps records of the users' presence in the system, provides the knowledge of which room an user is located. It is also responsible for transporting messages between clients.

D.3.1 Run the Server

To execute the server application, type the following line at the command prompt (Dos Prompt or Unix Shell):

```
java nameserver 4000
```

The number 4000 refers to the port that the client and the server use to communicate with each other.

D.3.2 Server Monitor

When the server application is up and running, there will be a window display showing information about the active users, their locations and the available rooms in the system (See Figure D-1). Each time a new user enters the system or a new room is created, the server monitor updates the corresponding list. The system administrator can use this information to monitor the status of the server. In addition, each room created in the system also has its own status window which keeps track of the users present the room.

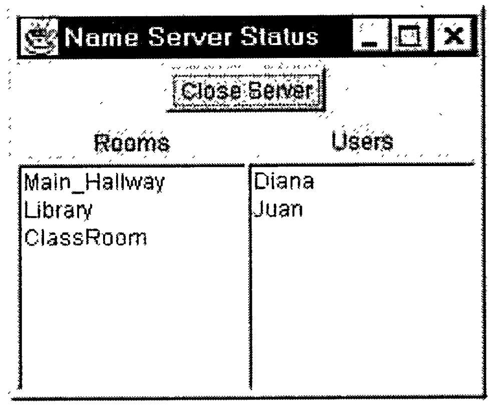


Figure D-1: Window of the Server Monitor

D.3.3 Close the Server

To bring down the Cliq! server, use the button labeled "Close Server" in the monitoring window shown in Figure D-1.

D.4 Execute the Application

To execute the program type the following line at the command prompt (MSDOS Prompt or Unix Shell):

java Cliq! <username> <password> <interval>[4]

When the system starts it will bring up a windows shown in Figure D-2. This window contains two buttons and a indicator light. The buttons permit the starting of the client application (use button labeled Start) and the exiting of the program (use button labeled Exit). The light indicator can be turned into three colors: yellow, green or red, each indicates a status of the network monitoring and a stage of casual contact. The Figure D-3 shows these possible stages and the color of the light associated to them. When the light is turned green the system indicates that the user is available to interact with other users. When the light is yellow, it indicates that there is at

[4]interval is the lapse of seconds that the system will run monitoring on the network port.

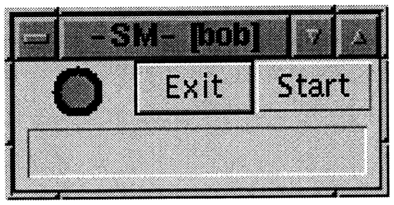


Figure D-2: Initial User Interface Window

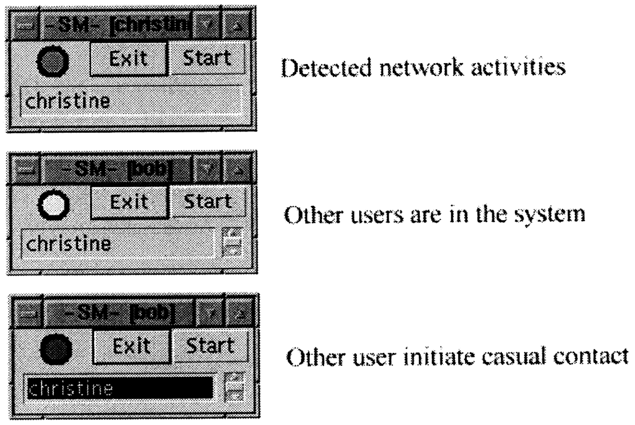

-SM- [christin
Exit
Start
christine
Detected network activities
-SM- [bob]
Exit
Start
christine
Other users are in the system
-SM- [bob]
Exit
Start
christine
Other user initiate casual contact

Figure D-3: Network Activities Status Monitor least one user in the system and a list of these users is shown. The light becomes red when other user request to interact with the passive user, the user that initiate the conversation will be highlighted in the user list.

D.4.1 Start a New Session

To start a new session in the system, press the button "Start" in the initial window (See Figure D-2). When the application starts, it comes in two windows. The first of them is the main window, where the users are located in a graphical room, the Main Hallway. Figure D-4 illustrates this main user interface. The second window is

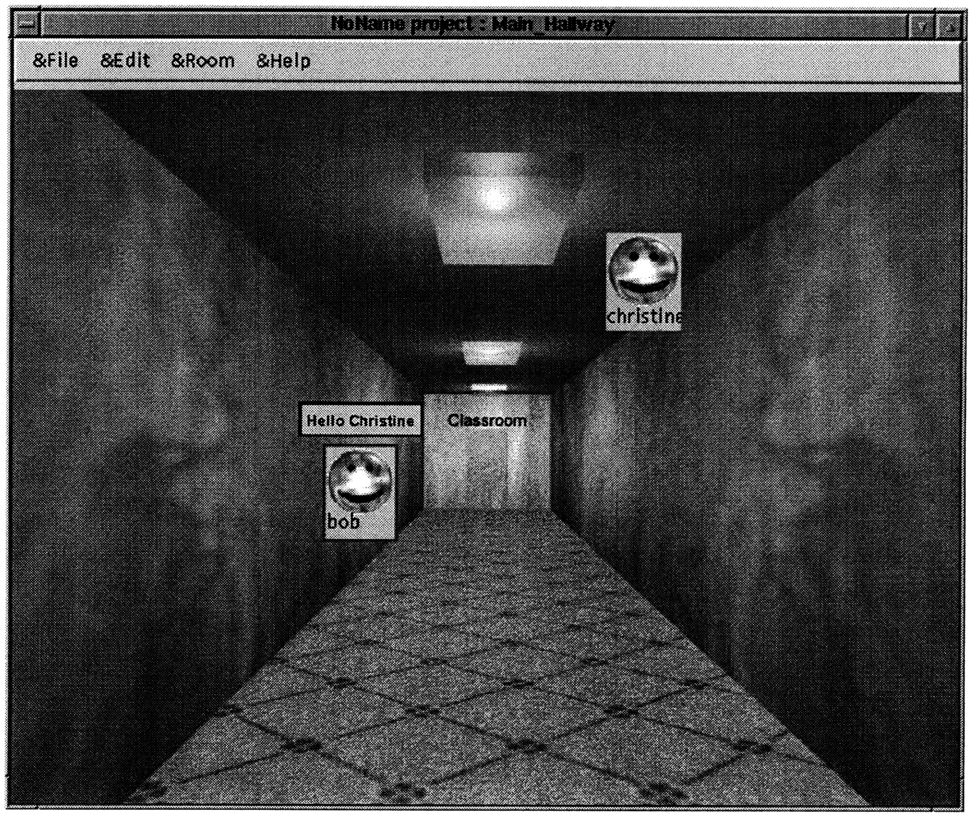


Figure D-4: Main User Interface Window of Cliq!

the toolbar interface for chat. Through this window, the user can communicate with other users through message passing and facial expression changing (See Figure D-5). The Cliq! system represents the users using avatars. The name of the user will appear at the bottom of the avatar iconic image to identify him/her. In addition, the avatar of the local user has a red border, this helps user to differentiate himself/herself from

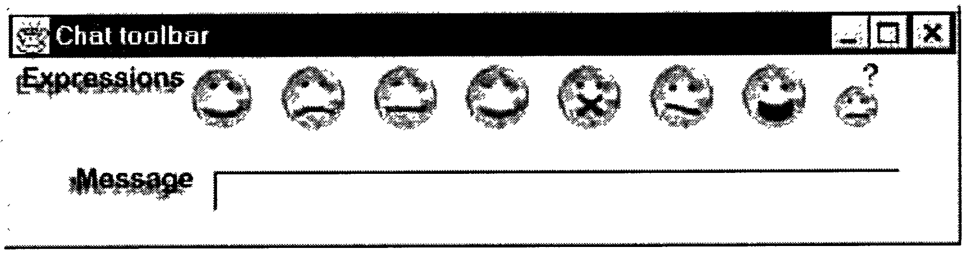


Figure D-5: Chat and Expression Toolbar

others. Figure D-4 shows the main window where two users are located. The avatar of user *Bob* has a red border around its icon image, indicates Bob is the local user.

D.4.2 User Navigation in a Room

To move the avatar inside a room just position the cursor above it. When the movement is allowed, the cursor will appear as a cross, at that moment, the user should press the left button of the mouse and drag the image to the desired position. One restriction is that, The user can only move his/her avatar and not the avatars of other users.

D.4.3 Change and Create a Room

The system is based on rooms where the users can explore and communicate with each other. When an user enters the system he/she fresh from the start, he/she is located in the main hallway, where he/she has access to other rooms. Each user can also create rooms, which can be public for all the others users or private. A private room means that only the user who creates the room has access to it.

Change of Room

The user can move from room to room. This action can be done in two ways: using the doors that are in the room or using the menu bar at the top of the main display. The room (except main hallway) has doors to other rooms. Figure D-6 shows the library, this room has a door to the main hallway at the bottom left of the window. To move from room to room using the doors can be done by simply placing the cursor over the door and left clicked on it. This operation will move the user to the corresponding room. Again, no doors are not available in the main hallway.
To move from room to room using the menu bar can be done by choosing the option "ChangeRoom" of the menu item "Room." Upon selecting this option the system will show a window that contains a list of the rooms to where the user could go. This

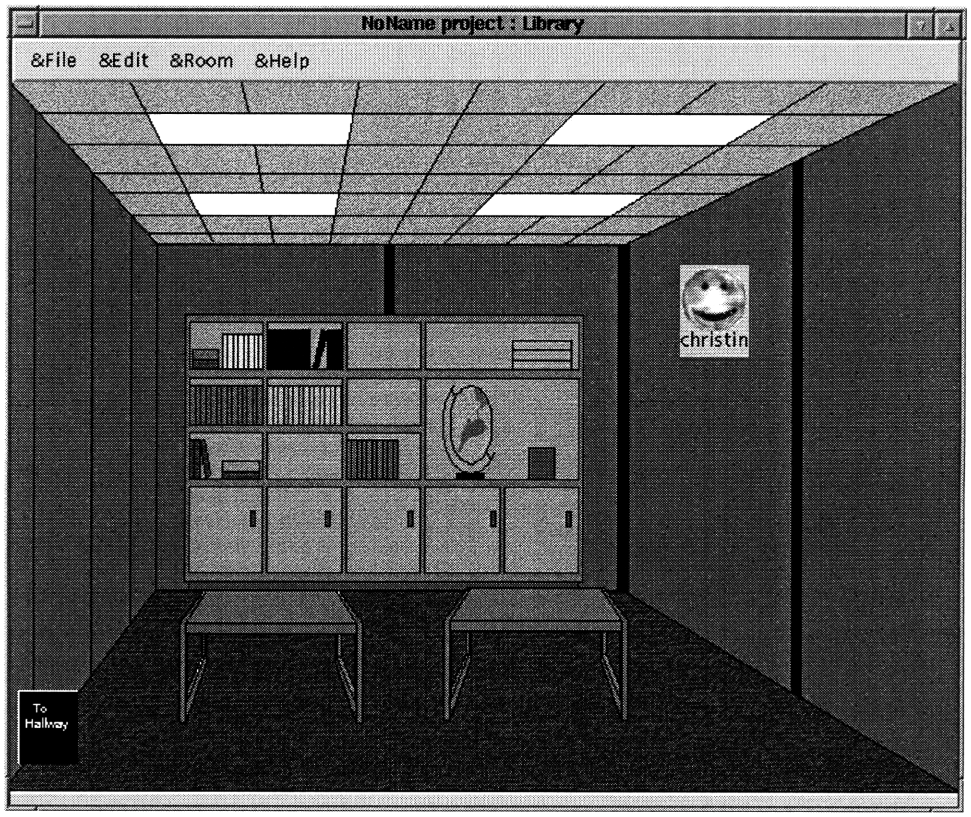


Figure D-6: Library with a Door to Main Hallway window is shown in Figure D-7. The user should select the room where he/she wants

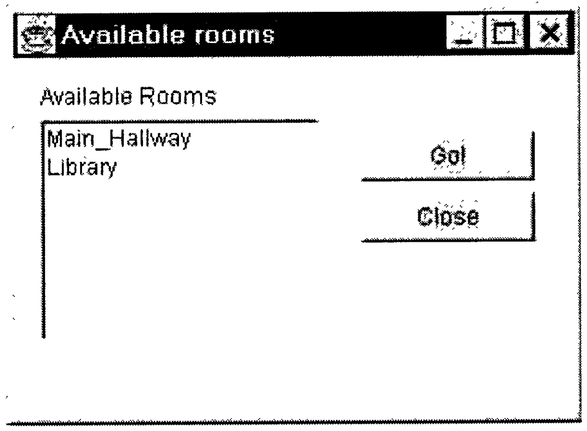


Figure D-7: List of Rooms where the User Can Go

to enter from the list of available rooms and then click on the button labeled with "GO!". The change will be done instantaneously. To cancel this operation click on button "Close".

Create a New Room

When the system starts the first time, there is only one available room, the main hallway. To create a new room, use the option "Create New Room" from the menu item "Room" in main display. The system will bring up a dialog box asking for the name of the new room (See Figure D-8). Once the room was created, the users can

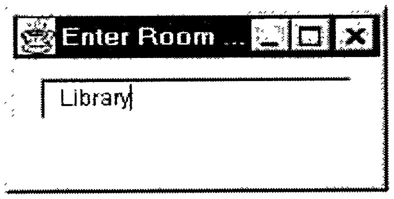


Figure D-8: Dialog Box to Introduce the Name of the New Room

access it using either the menu option ("change room") or the door indicators.

D.4.4 Send Messages

To send the messages the user must write the text in the field labeled "Messages" in the chat window (See Figure D-5). The messages are sent when user presses ¡Enter¿

key. After the message is sent it appear above of avatar representing the user who sent it. In Figure D-4 user Bob sent a message "Hi Christine" to the system.

D.4.5 Change of Expressions

The user can change his/her expression through two procedures. The first of them is using the expressions that are in the chat window. The second is by text messages that contain the shorthand of the emoticon expressions. The available expressions are:

- Smile
- Big Smile
- Sad
- Silent
- Blank
- Annoyed
- Wink
- Question

Change the Expressions Using the Chat Window

To change the expressions using the expressions from the toolbar (See Figure D-5), simply select the desired expression. This selection is made by pressing the left button from the mouse on the image of the expression. The selected image will appear right afterwards. Figure D-9 shows each figure and the expression that it represents.

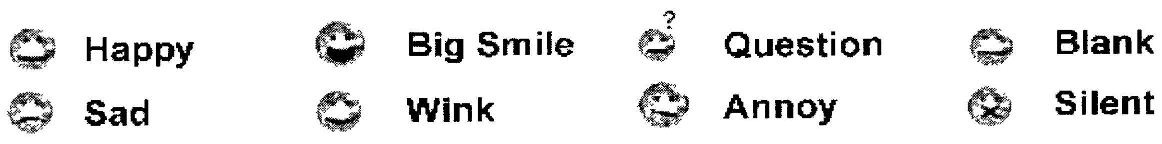


Figure D-9: Expressions in the Chat Window

Change the Expressions Using Messages

The second way to change the expression is embedded the emoticons shorthand in the text of a message. The emoticons are combinations of signs that represented expressions. The chat window has a field where the user could type in text. Once the text with the valid emoticon shorthand is written and the ¡ Enter¿ key is pressed the face or expression of the avatar is changed automatically. The valid expressions and their relationship with the images in toolbar window are shown in Figure D-10. When this option is used, the emoticon shorthand does not necessarily have to be

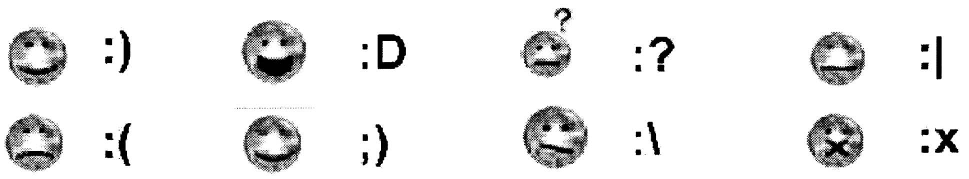
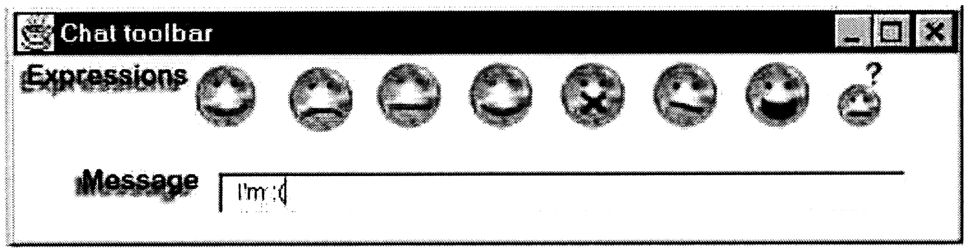


Figure D-10: Valid Emoticons and Their Relationship with the Expressions

typed alone. The user could introduce a longer message with the shorthand embedded in it (See Figure D-11).



I'm :(

Figure D-11: Emoticon Embedded in a Message

D.4.6 Logout

To exit the system use the option "Exit" at the menu item "File" in the main window. Once this option is executed, the application display is closed and the initial network monitoring windows will appear (See Figure D-1).

Bibliography

[1] H. Abelson and G. Sussman. *Structure and interpretation of Computer Programs.* MIT Press, Cambridge, MA, 1985.

[2] Penny Bauersfeld. *Software by Design: Creating People-Friendly Software.* M&T Books, New York, 1994.

[3] B. Boehm. A spiral model for software development and enhancement. *Computer*, 21(5):61–72, May 1988.

[4] Gary Booch. *Software Engineering with Ada.* Benjamin-Cummings, 1983.

[5] Gary Booch. *Object Oriented Design with Applications.* The Benjamine/Cummings Publishing Company, Inc., 1991.

[6] Gary Booch. *Object Oriented Design with Applications.* Benjamin/Cummings, RedWood City, California, second edition, 1994.

[7] F. Brooks. *The Mythical Man-Month.* Addison-Wesley, 1975.

[8] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Yourdon Press, Englewood Cliffs, New Jersey, second edition, 1991.

[9] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The Fusion Method.* Pretice-Hall, Englewood Cliffs, New Jersey, 1994.

[10] O. Dahl, E. Dijkstra, and C. A. R. Hoare. *Structured Programming.* Academic Press, London, England, 1972.

[11] P. Brereton (ed.). *Software Engineering Environments.* Wiley, 1988.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Desing Patterns: Element of Reusable Object-Oriented Software.* Addison-Wesley, Massachusetts, 1995.

[13] D. R. Garrison and D. Shale. Mapping the boundaries of distance education: Problems in defining the filed. *The American Journal of Distance Education*, 1987.

[14] T. Gilb. *Principle of Software Engineering Management.* Addison-Wesley, 1988.

[15] John Gundry. Understanding collaborative learning in networked organization. In A. R. Kaye, editor, *Collaborative Learning Through Computer Conferencing*, volume 90 of *Series F: Computer and Systems Sciences.* NATO, 1991.

[16] D. Ingalls. The smalltalk-76 programming system design and implementation. In *Proceeding of the Fifth Annual ACM Symposium on Principles of Programming Languages*, page 9. ACM.

[17] I. Jacobson, M. Christerson, P. jonsson, and G. Vergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* ACM Press/Addison-Wesley, Massachusetts, 1992.

[18] Ivar Jocabson. Basic use-case modeling. *Object Analysis and Design*, 1(2):15–19, 1994.

[19] Ivar Jocabson. Basic use-case modeling (continued). *Object Analysis and Design*, 1(3):7–9, 1994.

[20] D. H. Jonassen. Applications and limitations of hypertext technology for distance learning. In *Distance Learning Workshop*, San Antonio, Texas, 1992. Armstrong Laboratory.

[21] D. Keegan. *The Foundations of Distance Education.* Croom Helm, London, 1986.

[22] P. Naur and B. Randell (eds.). Software engineering: A report on a conference sponsored by nato science committee. Technical report, NATO, 1969.

[23] H. Perraton. A theory for distance education. In D. Stewart, D. Keegan, and B. Holmberg, editors, *Distance Education: International Perspective*, New York, 1988. Routledge.

[24] Roger S. Pressman. *Software Engineering, A Practitioner's Approach.* McGraw-Hill Book Company, Europe, fourth edition, 1994.

[25] J. Rumbaugh, M. Blaha, W. Premeriani, F. Eddy, and W Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[26] Linda Schamber. Delivery systems for distance education. In *ERIC Digest*, Denton, Texas, 1988. University of North Texas.

[27] Michael Waggoner. A case study approach to evaluation of computer conferencing. In A. R. Kaye, editor, *Collaborative Learning Through Computer Conferencing*, volume 90 of *Series F: Computer and Systems Sciences.* NATO, 1991.

[28] B. Webster. *Pitfalls of Object-Oriented Development.* M&T Books, New York, New York, 1995.

[29] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* PTR Prentice-Hall, Englewood Cliffs, New Jersey, 1991.