2000

Object-based, distributed online real-time communication system modeling and development.

Yongliang. Shao
University of Windsor

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Object-Based, Distributed Online Real-Time Communication System Modeling and Development

by

Yongliang Shao

A Thesis
Submitted to the College of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2000

Abstract

With the advent of the Internet and World Wide Web, online communication becomes increasingly popular. As broadband network technologies are becoming widely accessible, the media rich, highly interactive online applications for virtual enterprise, distance education etc. will be a reality. How to take full advantages of technologies available and effectively integrate them into a cohesive distributed distance education system that addresses adequately the scalability and the openness still pose a challenging question.

To address this need, we have developed an approach based on the principles of component-based development, utilizing standard component infrastructure and its services. We apply this approach to the online real-time communication system with capability of rich real-time interactions, which addresses key requirements for distance learning.

We provide a rigorous modeling and specification of the system, which describes the attributes that the distributed system must exhibit and prescribes the behavior of the system. This modeling shall allow software engineers to examine the behavior of systems under the development.

Based on this specification, we have developed a prototype of the distributed system to validate the effectiveness of our approach. The system is implemented with Java (JDK1.2) and the middleware is CORBA. The system is capable of handling multi-party real-time communication for a range of media types in a uniform manner. This system encompasses management facility such as authentication and allows dynamic services generation.

Running tests of this system in both Intranet and Internet environment have shown satisfactory results of this approach.

Keywords: Distributed Object, CORBA and CORBA Services, Online Communication, Modeling, Formal Specification, Distance Education, Asynchronous Decoupled Communication

To My Wife and My Family

Acknowledgments

I am extremely thankful to Dr. Indra A. Tjandra, my thesis supervisor, for his guidance, instruction and consistent support at every stage of this thesis work.

I am also very grateful to Dr. Yung H. Tsin and Dr. Nader G. Zamani for their valuable suggestions and comments on this thesis work. Dr. Jessica Chen should be thanked for her chairing my thesis defense.

Table of Contents

1. Introduction

This thesis is primarily concerned with online real-time communication modeling and development in connection with distance education. This introduction gives our motivations and objectives, introduces problems we will be tackling, as well as outlines the overall thesis organizations.

1.1 Motivations and Objectives

Today the Internet is so pervasive that it becomes a standard vehicle for delivering information and a very attractive medium for publishing. With the technology advances on the Internet and World Wide Web, online communication and collaboration have become increasingly popular. As broadband network technologies such as ATM (Asynchronous Transfer Mode), Cable Modem, and ISDN (Integrated Services Digital Network) are becoming common and widely accessible, the media rich, highly interactive online applications for virtual enterprise, distance education etc. will be a reality.

Distributed computing system will be the key to exploiting this astonishing high performance pervasive technology [Colou96]. Component based development will provide an effective approach being able to take full advantages of technologies available [Brown98]. Its underlying component infrastructure will provide a framework to seamlessly integrate distributed components being developed independently and running anywhere transparently in the network.

Our motivation is also largely driven by the online distance education system. How to take the full potential of technologies available to deliver distance education is one of the most exciting challenges for many educational institutions. Interaction is the key to successful education of any type. Unfortunately, a typical Internet-based distance learning system involves almost exclusively text (and/or graphics) and has insufficient capability of interactive communication with the instructors. For some online courses, the use of multimedia objects and real-time interactions are highly desirable [Ma98].

Despite the fact that a variety of online communication applications are available, how to effectively integrate them into a cohesive distributed distance education system that addresses adequately the scalability and openness still poses a challenging question.

To address this need, we propose an approach based upon the principles of component-based development, utilizing the standard distributed object infrastructure and standard available object services. Our open architecture will allow the distributed distance education system to extend at various ways, where new services can be added as required. We will formally specify this distributed communication system with mathematical modeling. By using formal or rigorous specifications, much of the ambiguity that is found inevitably in informal specifications could be eliminated [Alenc99]. It also enables software engineers to reason about and examine the behavior of the system under the development. Based on this specification, a prototype will be developed to validate the effectiveness of our approach.

Therefore our objectives in this thesis can be summarized as the follows:

- To develop an effective approach and methods for online real-time communication development. This approach will allow us to exploit fully the benefits that distributed systems can offer.
- To formally specify this distributed communication system for online distance education with rigorous mathematical modeling. This modeling will enable software engineers to reason about and examine the behavior of the system.
- To develop a prototype based on this specification to validate the effectiveness of the developed approach. This prototype shall have sufficient capacity and functionality for effective communication of a range of media types.

The online real-time communication system for distance education is actually a subsystem of the overall distance education system. The merit or the strength of this development approach also lies on the fact of its open architecture and infrastructure, which provides a foundation to seamlessly integrate distributed components and to extend for hosting new services.

1.2 Brief Overview on Methodologies

We build our distributed system with emphasizing its openness, scalable and extensible nature. The openness is a major attribute of distributed systems, which are largely constructed from open services built around a standard communication framework.

Components-based software development (CBSD or CBD) is about the creation and deployment of software-intensive systems assembled from components. It integrates the prefabricated component instead of building from scratch into the component architecture. Component-based technology is inseparable from the underlying infrastructures. In order to support a CBD approach, it is common to utilize some form of component infrastructure or component-oriented middleware to handle all of the complex details of component coordination.

We will discuses more detail about distributed system and component-based development in the following chapter.

CORBA (Common Object Request Broker Architecture) is one of component infrastructure for distributed object system. Its widely available object services and facilities will alleviate developer many concerns for distributed system development. CORBA is suitable for heterogeneous environment and supports different programming languages, which is an industry standard for distributed object technology.

One of important attributes in our approach is that we will utilize CORBA standard Event Service as the central hub for our real-time communications. This service is capable of performing administration work and conducting effective group cast. The genetic message type supported by the Event Service Channel will allow various typed messages to be communicated in a uniform manner.

Online real-time communication system among multi-parties can be effectively built with TCP/IP socket approach. Why we choose the object-based distributed approach? We conduct comparisons among these approaches in the following.

Compare with other approaches--socket, RPC and Java RMI:

Socket is an abstraction for inter-process communication across different machines. For a basic communication mechanism, Java supports sockets, which are flexible and sufficient

for general communication. However, sockets require the client and server to engage in applications-level protocols to encode and decode messages for exchange, and the design of such protocols is cumbersome and error-prone. Although hand optimized socket communication can outperform the systematic ORB approach, the cost for development, maintenance and management will well outweigh the gain of the performance. Such performance differences can only be noticeable in very high-speed networks where high performance communication is crucial [Gokha96].

Remote Procedure Call (RPC), which abstracts the communication interface to the level of a procedure call. Instead of working directly with sockets, the programmer has the illusion of calling a local procedure. RPC, however, does not translate well into distributed object systems, where communication between program-level objects residing in different address spaces is needed. However distributed object remote method invocation can be built on top of RPC. Additionally, many available CORBA object services will make the distributed object application much easier.

Java Remote Method Invocation (RMI) [Sun97] is a distributed object model for the Java language that retains the semantics of the Java object model, making distributed objects easy to implement and to use. Java RMI system assumes the homogeneous environment of the Java Virtual Machine, and the system can therefore take advantage of the Java object model whenever possible.

In essence both CORBA and Java RMI belong to distributed object paradigm, thus sharing many similarities. RMI is Java VM exclusive and optimized for Java environment, while COBAR is targeting to diverse heterogeneous environment, suitable for large-scale enterprise integration. CORBA has widely available infrastructure services, providing for administration, generic decoupled communication, transaction etc. To use these standard services instead of building from scratch is conformed to the principles of component-based development, which is pursued in this thesis.

1.3 The Organization of this Thesis

The remaining part of this thesis is organized as the following. In chapter 2, the background information related to this thesis will be presented. We will discuss major characteristics of distributed systems and fundamentals for component-based software development. They provide principles and guidelines for the generic distributed component system design. Then we will introduce basic concepts and the essential requirements for our targeted domain --Online Distance Education. In chapter 3, we will present CORBA overview and conduct investigations for a couple of Event Service implementations. CORBA is chosen as our middleware component infrastructure for our distributed system. Its basic Object Service, Event Service, is employed as our messaging central hub. By examining a couple of Event Service implementations, we will gain confidence in utilizing such service for our distributed system with desirable quality. In chapter 4, a mathematical modeling for our distributed system is presented. We will identify the essential components for our distributed communication system and communication patterns for online distance education. CCS (Calculus of Communication System) notation is used for our system modeling. This modeling will allow software engineers to examine the behaviors of the distributed system even before building the system. In chapter 5, detailed system design and implementation will be presented. We will firstly present the system development environment, system architecture, and major distributed component interfaces. Then we will discuss in detail about design and implementation, including components for chatting, drawing, images handling, video capturing, media playing, authentication, dynamic channel generation etc. Finally, in chapter 6, we will present conclusions and future works.

2. Background: Distributed System, Component-Based Development, and Our Targeted Domain--Distance Education

This chapter is primarily dealing with some background information, which provides principles and guidelines for the generic distributed component system design. We will discuss major characteristics of distributed system and fundamentals for component-based software development. They form a basis for our system design goals. We will apply these principles in the object-based online real-time distributed communication system. Although such design may apply to any distributed communication system in general, we have chosen our application domain for online education system. Towards this end, we will introduce some basic concepts and essential requirements for our targeted domain --Online Distance Education.

2.1 Basic Concepts of Distributed Systems

A distributed system consists of a collection of autonomous computers linked by a computer network and equipped with distributed system software [Coulo96]. Distributed system software enables computers to coordinate their activities and to share resources of the system—hardware, software and data [Silbe98]. Users of a well-designed distributed system should perceive a single, integrated computing facility event though it may be implemented by many different computers in different locations [Orfali96].

Distributed systems have become a norm for the organization of computing facilities. They can be used to implement general-purpose interactive computing systems and to support a wide range of commercial and industrial applications of computers. They are increasingly being used as the basis for new applications in areas as networked information services and multimedia applications, where communication is a basic requirement.

2.1.1 Major Characteristics of Distributed System

There are six key characteristics, resource sharing, openness, concurrency, scalability, fault tolerance and transparency, which are primarily responsible for the usefulness of distributed systems.

- **Resource sharing**: The term resource here is a rather abstract one, including from hardware components such as disks and printers to software-defined entities such as files, windows and database. Resources in a distributed system are physically encapsulated within one of the computers and can only be accessed from other computers by communications. For effective sharing, each resource must managed by a software program—resource manager, which offers a communication interface enabling the resource to be accessed, manipulated and updated reliably and consistently.
- **Openness:** The openness is the characteristic that determines whether the system can be extended in various ways. The openness of distributed system is determined primarily by the degree to which new resource-sharing services can be added without disruption to or duplication of existing services.

Open distributed systems are based on the provision of a uniform interprocess communication mechanism and published interfaces for access to shared resources. Open distributed system can be constructed from heterogeneous hardware and software, possibly from different vendors.

- **Concurrency**: Concurrency brings benefit of higher performance. Concurrency and parallel execution is natural to distributed systems in terms of the following facts: the separate activities of users, the independence of resources and the location of server processes in separate computers. Concurrent accesses and updates to shared resources must be synchronized to ensure that they do not conflict.
- **Scalability**: Scalability is that the system and application software should not need to change when the scale of the system increases. Scalability has been a dominant concern in distributed systems during the last decade, and its importance continues. The demand for scalability in distributed systems has resulted in such a design philosophy—no single resource (hardware or software) is assumed to be in restricted supply. As the demand for a resource grows, it should be possible to extend the systems to meet it. Some techniques

such as replicated data, caching and deployment of multiple servers, have been successfully applied in coping with large-scale applications.

- **Fault tolerance**: Computer system sometimes may fail due to the hardware or software. In this case, programs may produce incorrect results or they may stop before they have completed the intended computation.

Generally there are two approaches for designing of a fault-tolerant computer system, 1) hardware redundancy: the use of redundant components and 2) software recovery: the design of programs to recover from the faults.

The servers can be designed to detect faults in their peers; when a fault is detected in one server, clients are redirected to the remaining servers. Distributed systems provide a high degree of availability in case of hardware faults. When one of the components in a distributed system fails, only the work using the failed component is affected. A user may move to another workstation and a server can be started on another computer.

- **Transparency**: Transparency addresses the needs of users and application programmers to perceive a collection of networked computers as an integrated system, hiding the distributed nature of the resource used to perform the user's task. The implications of transparency are major influences on the design of the system software.

The two most important transparencies are access transparency—enables local and remote information objects to be accessed using identical operations, and location transparency—enables information objects to be accessed without the knowledge of their location. They are sometimes referred to together as network transparency. Other transparencies include Concurrency, Replication, Failure, and Migration transparencies, etc. They all together provide a useful summary of the motivation and goals for distributed system.

It should be noted that the above characteristics are not automatic consequences of the distribution. Distributed system and application must be carefully designed in order to ensure that these key characteristics are attained.

2.1.2 Basic models of Distributed Systems

There are two basic models, i.e. the client-server model and the object-based model for distributed systems.

The client-server model is a well-known and widely adopted system model for distributed systems. In this model, there is a set of server processes, each acting as a resource manager for a collection of resources of a given type, and a collection of client processes, each performing a task that requires access to some shared hardware or software resources. Some processes can be both client and server processes. This is the case when resource managers may themselves need to access shared-resources managed by another process. The basic scenario in this model is as follows: Client processes issue requests to servers whenever they need to access one of their resources. If the request is a valid one, then the server performs the requested action and sends a reply to the client process.

The object-based model is similar to the traditional object-oriented programming model in which every entity in a running program is viewed as an object with message-handling interfaces providing access to its operation [Myer88]. In the object-based model, each shared resource is view as an object. Objects are uniquely identified and may be moved anywhere in the network without changing their identity. The basic scenario in this model is as follows: A resource-using program sends a message containing a request to the corresponding object whenever it needs to access a resource. The message is dispatched to the appropriate process that performs the requested operation and sends a reply message to the requesting process if required.

The object model is attractive due to its simplicity and flexibility. It enables all shared resources to be viewed in a uniform way by the resource users. Distributed object paradigm is the extension of the traditional object oriented notion, in the sense that objects can interact with other objects that are not within their address space in a same manner. This thesis is embracing object-based model. The research here is centering on methodologies for object-based distributed system.

2.2 Component-Based Development

Component-based development (CBD) is an essential approach adopted in this thesis for software system development. This is the trend of current software development. In this section, we will present some fundamentals related to CBD. We will discuss what its benefits are and the technologies that will make this approach a reality.

2.2.1 The Trend of Software Development

Recently, component and component-based software engineering have gained substantial interest in the software community [Kozac98, Brown96]. During the last decade the focus of business software applications has shifted from centralized and monolithic applications to the modular (component-based) and distributed systems [Brown98, Jacob97, Adler95]. This trend underscores some clear advantages of a modular and distributed system, which easily adapts to the needs of the market; does not rely on the particular technology or single vendor; engages in high level software reuse; and is characterized by high scalability and extensibility [Butle99]. Using prefabricated components to assemble new applications will dramatically shorten the software development life cycle [Dean97]. These advantages will lend companies a competitive edge in this ever-dynamic market.
Software components are basically reusable software. In the 1980s software reuse for rapid software development was pursued by broad application domains [Jacob97]. They tried to gather application assets, and wait for the flood of customers to make use of them. However, the majority of "reuse initiatives" did not succeed [Brown99]. There were a number of obvious reasons: the technical infrastructure supporting reuse was immature, cataloging assets was hard, the assets were diverse and of varying quality, the interfaces and behavior of assets were poorly defined, etc. Why CBSE may be more effective now? The reasons may be the following: the maturing object technology; object oriented programming has better structure for facilitating CBSE; domain-specific libraries and frameworks starting to appear; potential robust technology support; the maturing web infrastructure [Brown99].

The emergence of CBD is one of the most important events in the evolution of information technology [Butler99]. While developments in hardware and networking have provided increasing capability, software development has remained a 'craft' industry, often with problems of delayed and canceled projects, inadequate quality, long cycle times, and high costs. Software packages can not be viewed as an adequate alternative either because they are often involving long, costly, and difficult implementation projects, and even more difficulties in integration and upgrading.

2.2.2 What Component-Based Development Can Offer

There are economical reasons, such as reducing cost, shortening time to the market, speeding up technology refinement and improving interoperability, which drive many organizations towards greater use of available commercial solutions. Consequently, many application systems consist of a combination of COTS (Commercial Off the Shelf) packages, legacy data, home built solutions and integration code. The only way to design, assemble, and maintain these application systems is to consider them to be collections of parts modeled as interacting components.

Additionally, the style and architecture of applications being developed has significantly shifted from centralized mainframe-based applications accessed via terminals over proprietary networks to distributed, multi-tier applications remotely accessible from a variety of client machines over Intranets and the Internet. Instead of building a small number of large projects, the organizations now are typically building a large number of smaller projects.

From technical point of view, the main advantages of adopting a component-based approach to overall development are [DSouz98]:

- Reuse of implementation and related interfaces at medium-granularity, which is higher level reuse than that of objects.
- Unit of maintenance and upgrade: no longer need to upgrade entire system, components get replaced or added as needed.
- Parallel development: identifying the medium grain chunks, and focusing on early design of interfaces makes it easier to develop and evolve parts in parallel.

- Scalable: Interface-centric design yields scalable and extensible architecture.
- Leverage standards: Since component technology implies that some base set of standards for infrastructure services are provided, a large application can leverage off these standards and save considerable efforts.
- Manageable and self contained units: A component is technically practical unit to configure, version and package.
- Higher-level capabilities: It can support capabilities that are impractical for "small" object such as language independent access of interface and transparent interaction between distributed components.

2.2.3 Component Infrastructure Technologies

Component-based technology is inseparable from the underlying infrastructures. In order to support a CBD approach, it is common to utilize some form of component infrastructure or component-oriented middleware to handle all of the complex details of component coordination. In essence, this infrastructure provides a common set of component management services available to all of the components interested in using that infrastructure.

There are three major component infrastructure technologies available now, including OMG's CORBA [OMG95, OMG98b], Microsoft's COM/DCOM [Micro96], and Sun's JavaBeans/EJBs [Sun98]. DCOM provides suitable solutions for Microsoft environment and EJBs is ideal for Java solution. CORBA has clear advantages in terms of crossing platform and language boundaries [Meta98, Chung98], which is truly suitable for large-scale enterprise solution. We will discuss CORBA in detail in next chapter.

As about our campus environment, there are machines of UNIX/LINUX, Windows (NT and Win95/98), Macintosh etc., which has certain diversities already. CORBA would be a good choice for the infrastructure technology to cope with this devise heterogeneous environment.

2.3 Online Distance Education System

In this modern fast paced society pushed and pulled by an ever-expanding technology, people consistently have to update or reengineer their knowledge and skills. Education is becoming one of most important enterprises in this century. As we will be building distributed system for online distance education, we must understand the context and requirements of the system.

2.3.1 Context of Online Distributed Education

It is estimated that at least half a million Canadians are participating in about 4000 distance courses across Canada. There are over 800 U.S. universities and colleges offering degree courses online, more on non-degree courses according to Peterson's, the college guide publisher. People believe that there is so much need to provide time-flexible and place-flexible education that is of high quality. Many people choose distance education rather than the traditional classroom setting due to various reasons, such as time, distance, physical disabilities, transportation limitations and expenses, or non-school commitment [Schne94]. In addition, even for traditional full time students, distance learning can provide attractive options with great flexibility.

When the Internet and World Wide Web (Web) took off in 1994, many schools and researchers used the Web to construct a distance education environment. The Internet has become a standard vehicle for delivering information and a very attractive medium for publishing. How to take the full potential of the technology available to deliver distance education is one of the most exciting challenges for many educational institutions.

Interaction is the key to successful education of any type. Unfortunately, a typical Internet-based Distance Learning System involves almost exclusively text (and/or graphics) and has insufficient capability of interactive communication with the instructors. For some online courses (for example chemistry, physics, mathematics, geography, engineering and computer science, just to name a few), the use of multimedia objects and real-time interaction are desirable.

Although the bandwidth of the Internet itself is often the bottleneck for the delivery, the real-time media (video and audio) over the Internet has recently come to the reality with minimal speed requirement of 28.8kbps modem. The current stage of communication

technology, such as the availability of ISDN (Integrated Service Data Network) and ATM (Asynchronous Transfer Mode), enable us to develop more advanced distance education system relying on broadband computer networks. Consequently, it will be realistic in the near future to use multimedia and to allow real-time interaction between students and instructors.

2.3.2 The Trend in Distance Education

There are many buzz words around in respect to distance learning or online education, such as Virtual Classroom, Cyber School, Wired Education, Education on Demand, TeleLecture, just to name a few. Traditionally, universities are involved with research, teaching and service. Online education is concerned with content, context (lifestyle), and certification. Education in the 21st century will simply be considered as education, regardless of where, when, and how it is delivered.

Because of the capability of the Internet to support discussion groups and text, graphics, audio, video, file transfers over electronic mail, in asynchronous format, and also video conferencing, whiteboards, and chat in synchronous real-time modes, a different learning medium has evolved in contrast to the traditional style. Since Web-based instruction is such a new medium, evidence of the effectiveness of online courses compared to traditional instruction is lacking. Therefore Web-based instruction must be designed to accommodate individual learning styles. This does not mean using all available technologies but instead using those appropriate technology mechanisms that will directly contribute to enhance learning.

To build a quality online education system is not quick, not easy and not cheap in the short term. It has a value-added perspective, such as increasing access, serving rural communities, and expanding student education choices. The challenge for the future will be to balance access, quality, and cost in delivering courses and programs at a distance.

2.3.3 Requirements for Online Distance Education

As an online learning system, it must possess some certain features and functionality in general. The system should have an open architecture design so that it can adapt to the new technologies to take their advantages. Since the online learning systems have been developed for a number of years, people have certain criteria about what to look for in such systems although they may not agree with each other for every detail. Sunil Hazari [Hazar98] presented an exhaustive list regarding this criterion. In essence, we can classify requirements of online education into the following categories: (1) Static, dynamic documents and asynchronous interactions (e.g. static web page management, dynamic document with CGI and front end scripting or client-server applications, email, message board and student record management). (2) Interactive teaching: active, synchronous and real-time (e.g. text chat with white boarding—facilitating or simulating computer lab instructions, streaming audio/video/animation) (3) Active/interactive learning (remote application invocation—compile and execution, self-evaluation, quizzes, assessment and so on).

While the first category consists of basic features that system must exhibit, we will emphasize the importance of interactive teaching and active/interactive learning in our distance learning system. This is because the interaction is the key for any successful education. We will develop a system enabling interactive features, which shall bring classroom-like and lab tutorial experiences to the online education. In particular, multiparty of students and instructors can interact in a real-time. The instructor can send images captured lively or pre-stored to the participants, while they can have live chat and sketch their ideas with collaborated drawings.

Broadly speaking, interactions can be either between the students and instructors in real-time or between student and computing processes (or services). The above intended built system is primarily addressing the need for the human online real-time interactions. As about the human and process interactions, they could be remote compilation or execution for instance in Computer Science education. Our open component-based architecture will enable these sub systems to be integrated seamlessly into the overall online education system. In fact our generic architecture (see Figure 5.1 for detail) is very inclusive, which

encompass all essential requirements from all three categories with emphasis on the real-time online communication capabilities.

3. CORBA Overview and its Event Service Investigation

OMG's CORBA [OMG95, Siege96] is an industry standard for middleware or distributed component infrastructure, which integrates seamlessly the independent developed components running over a heterogeneous environment. It has been chosen for the middleware glue for our object-based distributed online communication system.

CORBA Event Service is one of OMG's common object services [OMG98b], which facilitates decoupled and asynchronous communications style among CORBA objects. The Event Service will be a central hub for our message communication system. To gain significant confidence in utilizing this service, we will investigate a few examples of the Event Service implementations. By carefully examining the system attributes among different implementations, we feel that it is very desirable to have a precise mathematical model to describe the system attributes or quality of service (QoS) that must be exhibit in the corresponding implementations.

3.1 Overview of CORBA

CORBA — Common Object Request Broker Architecture — was designed by the Object Management Group (OMG) to support open distributed communication between objects across a wide variety of platforms and languages [OMG95]. Interestingly, despite the "Object" in its name, CORBA does not directly expose the notion of object identity; and could more properly be considered as a distributed component framework [Szype98]. In fact, a "Distributed object is a component by definition" [Orfal96].

To meet its goals of heterogeneous computing, CORBA opted to become a source-code standard, rather than a binary one. Component interfaces are defined within modules, using the CORBA IDL (Interface Definition Language). Different programming languages have standardized bindings to the IDL. Programmers either (a) manually write IDL code, then compile it into the source-code versions needed to write their implementations; or (b) use a vendors programming language compiler that offers direct generation of IDL.

3.1.1 OMA's Reference Architecture

Figure 3.1 illustrates the OMA (Object Management Architecture) reference architecture. The OMA contains the following:

• CORBA "bus"--ORB: the base level of IDL-based interface definitions, the interface and server repositories, and the request broker.

• CORBA Services: a variety of largely infrastructure services, ranging from events, transactions, relationship, naming, lifecycle, licensing, and externalization.

• CORBA Facilities (horizontal): A set of higher-level object specifications providing commonly required services to applications such as printing, email, compound documents, structured storage, workflow, etc.

• Domain Interface (vertical): standards for business objects in "vertical" domains, including health care, telecomm, financials, etc.

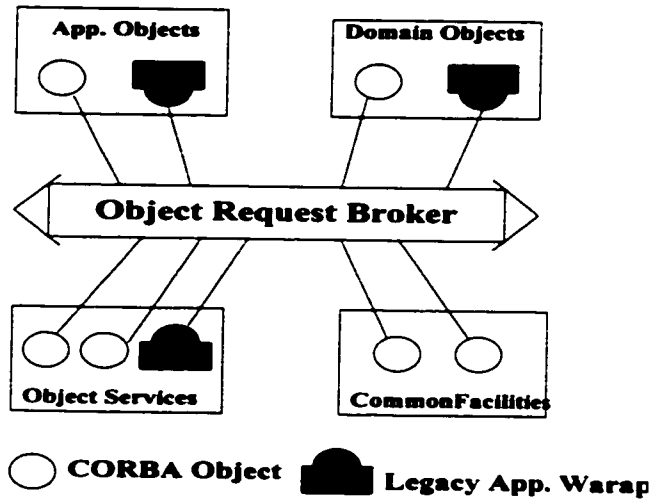

Figure 3.1 OMA reference Architecture

There are of course your application specific objects as well. In the figure, the Objects and Legacy Application Wrappers are represented as different shapes, to highlight the capability of CORBA in incorporating legacy code.

Almost all major programming languages have their mapping to CORBA IDL. They include C, C++, SmallTalk, COBOL and recently Java [Siege96, Vogel97].

3.1.2 CORBA Essential Components:

Figure 3.2 shows the CORBA detail architecture with essential components. We will describe the main features for CORBA, which are associated with components in this figure.

An ORB agent or daemon process: It is responsible for locating and launching servers and facilitating client communication with servers.

IDL stubs: The code generated by IDL compiler for specific IDL interface to allow static invocation of operations of that interface via proxy objects.

DII—Dynamic Invocation Interface: This acts like generic client stub, which enables clients to make operation invocations even without IDL stubs.

IDL skeleton: The code generated for a specific IDL interface that invokes object implementations of that type.

DSI--Dynamic Skeleton Interface: The counterpart of DII at server side. A generic interface that allows interpretation of incoming requests to a server for IDL types that are not known at compiler time.

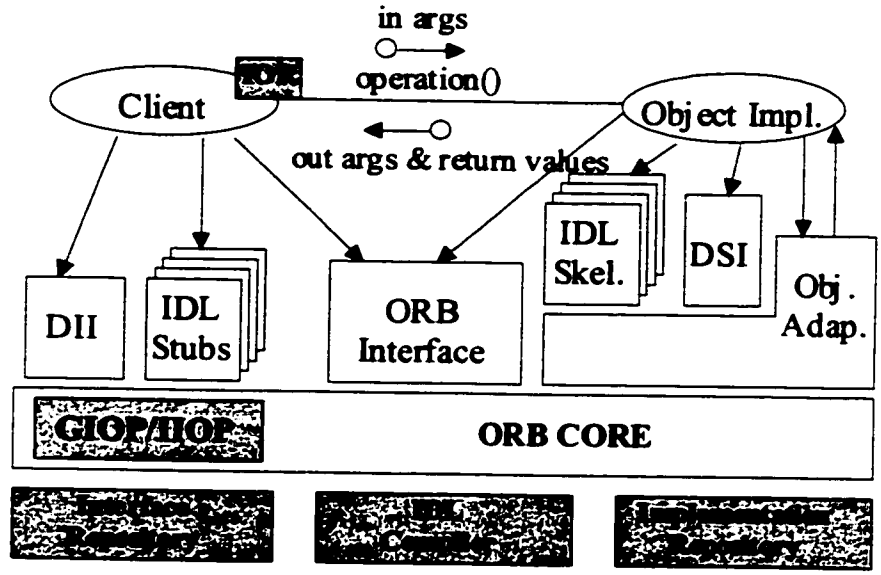

Figure 3.2 CORBA architecture, showing essential components.

Object Adapter: This is the component capable of activating servers whose objects are required by invocations. BOA is also the place to specify the thread policy, i.e. thread per client, thread per invocation etc.

CORBA GIOP/IIOP: CORBA 2.0 [OMG98a] includes specification for ORB interoperability called General Inter-ORB Protocol. This is a description of the way in which IDL types are laid out in a message format for invocation request. This GIOP message format is the basis for the specification of the Internet Inter_ORB Protocol (IIOP), which uses TCP/IP as its transport protocol. Any CORBA 2.0 compliance must implement IIOP.

An optional additional protocol, DCE Common Inter-ORB Protocol (DCE_CIOP) is specified, which utilizes DCE-RPC mechanism [DCE95].

CORBA2.0 also provides a specification for Interoperable Object Reference (IORs), which allows an object reference from any compliant ORB to be used by a client using any other ORB. IIOP ensures the interoperability among different ORB vendors.

3.1.3 Location and Language Transparencies in CORBA:

As discussed in chapter 2, transparencies are major design issues in distributed systems. How does CORBA achieve the location transparency? When the client obtains the remote object reference (ObjRef in Figure 3.2), the client can just invoke the remote object operation the same way as it invokes the local one (same address space). In fact, the operation call will go through the following steps. 1) The IDL stub will marshal the passing parameter as necessary. 2) The ORB will locate the remote object and send the function call through the transport and network layer. 3) The server side ORB will receive the call, together with BOA, identify and activate the corresponding object implementation servant. 4) IDL skeleton will be used to unmarshal the calling parameters, and the operation associated with the object servant will be invoked. If there are any return values, the return message will be going through the reversed way. In fact the processes described above encompass many other transparencies, such as transport, network and operating system transparencies.

One of other transparencies that assists in the integration of distributed applications in CORBA is programming language transparency. This is due to the fact that CORBA objects are typed by interface definitions and referenced by object references, the implementation behind the interface can be in one of several programming languages (SmallTalk, C++ or Java etc.) If the implementation language is Java, a new transparency --implementation transparency will be introduced. This is due to the fact of Java "Write Once Running Anywhere", platform independent nature.

3.2 CORBA Event Models and Event Channel

CORBA Event Service is one of OMG's common object services, which facilitates decoupled and asynchronous communications style among CORBA objects [Schmi97, Schmi97a]. As you may have noticed, the message sending between CORBA object is essentially synchronous. Although "one way" semantics is basically asynchronous invocation, but various implementations of "one way" makes it inadequate to be an alternative of asynchronous communication.

CORBA event supports both push and pull models. There will be push/pull suppliers and push/pull consumers. CORBA event service utilizes the so-called event channels as a mediator to decouple the event suppliers and consumers. The push event supplier will push the event to the event channel (the event for pull supplier will be pulled by the event channel), the events in the event channels will be consumed by the interesting parties via the way of either pull or push. There is an event queue management in the Event Channel, hence providing a buffer for the decoupling.

Figure3.3 illustrates the Event Services, showing various event models—Push/Pull Server and Push/Pull Client. There could be multiple instances or none for each event model to connect to the Event Channel. Event Service provides all kinds administration work for registered parties, thus free developers' concerns to manage those connections.

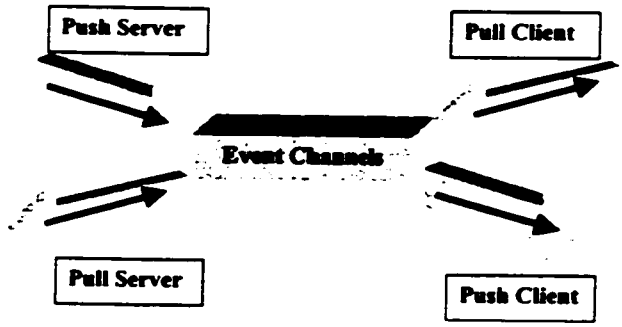

Figure 3.3 CORBA Event Services, showing the Push Server, Pull Server, Pull Client and Push Client. The 3-D look arrows and the solid arrows correspond to the direction of the control or invocation flow and the data/message flow respectively.

The role of the Event Channel as a function of communication models can be summarized at the following table. Based on the nature of our communication system, we will primarily employ the Push/Push model—Notifier. This stems from the fact that we need to propagate the communication messages as soon as possible to the interested parties.

		Supplier	
		Push	Pull
Consumer	Push	Notifier (Canonical Push/Push Model)	Agent (Hybrid Pull/Push Model)
	Pull	Queue (Hybrid Push/Pull Model)	Procurer (Canonical Pull/Pull Model)

Table I. The role of the Event Channel as a function of communication models

3.2.1 Event Service Specification: IDL Interfaces

The interfaces of the two primary components in Event Service have been defined with IDL in the modules of CosEventComm and CosEventChannelAdim. The following IDL code segments are part of these specifications. This will be served as a starting point for the implementation discussion in the following sections. For thorough specifications, please refer to chapter 5 or CORBA Service Specification [OMG98b].

```
module CosEventComm{
        exception Disconnected{};
        interface PushConsumer{
                void push(in Any data) raises (Disconnected);
                void disconnect_push_consumer();
        }
        ...
        ...
}
module CosEventChannelAdmin{
    exception AlreadyConnected{};
    ...
    interface ProxyPushConsumer:CosEventComm::PushConsummer{
        void connect_push_supplier(in CosEventComm::PushSupplier push_supplier)
        raises(AlreadyConnected);
    }
    ...
}
```

3.3 Rational for Evaluations of Various COSEvent Implementations

OMG recognizes that application domains have diverse requirements for the event style communication. Such diversity includes the requirements for reliability, from the best effort to guaranteed delivery semantics, performance, availability, throughput and scalability [OMG98b].

There exists no such one-fits-all implementation that can optimize such diverse technical requirements. OMG intended to define the QoS very vaguely so as to encourage innovations for different requirements. This also poses a problem on selecting an adequate Event Service implementation for specific application logic. Therefore it is imperative to have a mathematical model which is capable of addressing precisely the system attributes and behaviors.

There are many COS (CORBA Object Service) Event implementation available, being provided by different vendors. Examples are OOC's ORBacus Event [OOC98, OOC99b], IONA's OrbixTalk [Orbix98], and HP ORB Plus Event Service (based on IIOP). Because we have the source code available for ORBacus, we will primarily focus on the investigation of this product. There are many details in the implementation, which can have significant impact on the system properties. However, this kind of impact can not be thoroughly realized through examining solely the manual or software documentation. This suggests the need for looking into the code of the implementation. OOC has made its source code available. Moreover, we have ORBacus Version 3.1.1[OOC98] and Version 3.1.3[OOC99b] for a comparison. More interestingly, these two versions provide a sharp contrast in terms of the degree of concurrency, which could be critical in a distributed environment.

3.4 Examining ORBacus Event Service V3.1.1 Implementation

ORBacus 3.1.1 event service is kind of primitive if we compare it with its 3.1.3 version. It is unfair to compare an early product with the relatively matured one. However, the point here is to shed the light upon the quality of services affected by the implementation. It is not intended to evaluate the sophistication associated with both implementations.

There are eight classes in the package of com.ooc.CosEventChannelAdmin, i.e. EventChannel, SupplierAdmin, ConsumerAdmin, ProxyPushConsumer, ProxyPullConsumer, ProxyPushSupplier, ProxyPullSupplier, and Server. The first seven classes are the implementations corresponding precisely to the IDL interfaces defined in org.omg.CosEventChannelAdmin module. The Server is responsible to start Event Channel Service. It is worth noting that the concurrency thread model of the BOA (Basic Object Adapter) for this Server has been specified as ThreadPerClient. The option to be able to specify the particular thread model for the BOA handling requests has been an important feature for ORBacus.

The suppliers and consumers follow the same pattern as the general case to register themselves to the SupplierAdmin and ConsumerAdmin respectively. For example, a

PushConsumer obtains the IOR of the ConsumerAdmin by invoking the method for_consumers of the EventChannel. Further the ConsumerAdmin will create a ProxyPushSupplier object, which exchanges its object reference with that of the PushConsumer. Perhaps most of implementations will follow this pattern for the registration. This is because a well-defined protocol bounded by the IDL interfaces. Because this protocol involves double hand shaking (additional registration to the Consumer/SuplierAdmin thus enabling the graceful disconnection management), it is sometimes considered as an overkill registration for certain circumstances [Schmi97b]. However this feature is necessary if Event Channel is designed to be used in general cases.

A sequence diagram for the event propagation process utilizing ORBacus3.1.1 is illustrated in Figure 3.4. The diagram is for the case when there is one PushSupplier, two PushConsumers and one PullConsumer. However these remote objects (in terms of the view of Event Channel) are not presented in the diagram due to the space limitation. The scenario is the following: 0) The PushSupplier pushes an event any to its corresponding ProxyPushConsumer. 1) This ProxyPushConsumer invokes EventChannel.receive(any). 2) EventChannel in turn invokes the synchronized method ConsumerAdmin.receive(any), note that this synchronization is an important concurrency control mechanism when multi PushSupplier may present. This will ensure the consistent order of the events for all consumers. 3),4),5) The ConsumerAdmin will further push the event to the all registered ProxySuppliers one by one. It is important to notice that in this implementation each ProxyPushSuplier will immediately invoke its corresponding remote PushClient within this nested call. However, in the case of ProxyPullSupplier, the event will be stored within the queue locally managed by the ProxyPullSupplier. In this situation, the nested calls can be returned quickly without involving an additional remote method invocation.

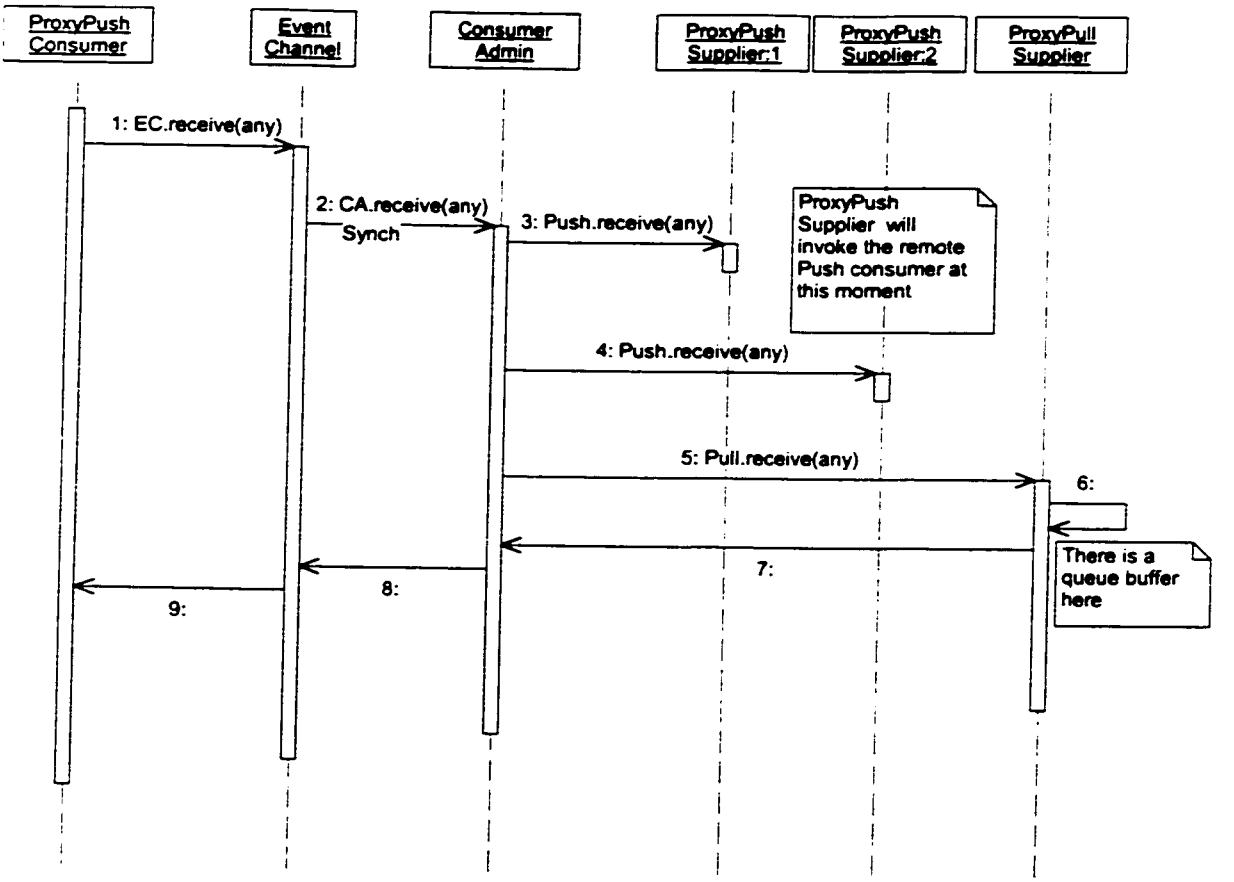

Figure 3.4 Sequential diagram for the event propagation for ORBacus 3.1.1

It is not difficult to infer that this implementation is inadequate for the circumstance of one-to-many or many-to-many push-push (notification semantics) communication style. Each supplier has to wait for that which all PushConsumers have received the event. During this period of time, this supplier will be blocked for the next move. For a 1-to-m push-push communication, the supplier will be waiting for 1+m remote calls before its regaining the control. Perhaps in case of one-to-one push-push communication, this implementation is still useful. If that is the case, it is better to engage the supplier and consumer directly without employing the event channel in the first place. If there are more than one supplier, more trouble will be warranted. All the ProxyPushConsumers (corresponding to the Suppliers) will be waiting for the synchronized call to the ConsumerAdmin.receive(any). Therefore in terms of performance and concurrency access, this implementation is incompetent to accomplish push-push style communication. On the other hand, one may notice that the ProxyPullSipplier will

maintain a queued event. This is the only type of the queue managed within the Event Channel. Therefore this implementation is still useful for the push-pull (queuing semantics) style communications. For instance, the assembly line pattern implemented with Event Channel Service (push-pull) is a successful example of using this implementation [Tjand99]. There are some other issues related to the fairness and potential starvation. Because these issues are common to the implementations of the both versions, we will discuss them in the following section.

3.5 Examining ORBacus Event Service V3.1.3 Implementation

ORBacus Event 3.1.3 is becoming very sophisticated comparing to its previous counterpart. It was claimed to be one of the most comprehensive and complete implementation of OMG's COSEvent. There are three packages CosEventChannelAdmin, CosEventServer, CosTypedEventChannelAdmin and about 30 classes in total. Noticeably, Typed Event Channel has been implemented in this version. In order to compare with the conclusion we arrived in the last section, we will still concentrate on issues related to the event propagation through the event channel.

In essence, the process before the ProxySuppliers[1] remains unchanged as far as the event propagation concerned. The fundamental change in the ProxyPushSupliers is that it now maintains an event queue (actually managed by PusherbasedThread). The ProxyPushSupplier is now a multi-threaded accessed entity. One thread is responsible to push the event to the remote consumer when there are events in its event queue. Another thread is passively waiting for its event queue to be inserted by the ConsumerAdmin. The wait() and notify() style synchronization has been extensively used to improve the concurrency and to reduce resource consumption.

The sequence diagram will be extremely complicated due to the many synchronized threads. Therefore we will rely on the simple diagram (space diagram) as shown in Figure3.5 to assist the description of the whole propagation process. We want to emphasize that there are individual queues for each ProxySuppliers, which will decouple

[1] ProxySuppliers indicates ProxyPushSupplier or ProxyPullSupplier or both; the same to ProxyConsumers

the remote push() invocation from the ConsumerAdmin.receive(). The actual implementation for such a queue management is rather complicated, spanning to a few threaded objects (not shown in the diagram). The presentation of such detail is beyond our current interest.

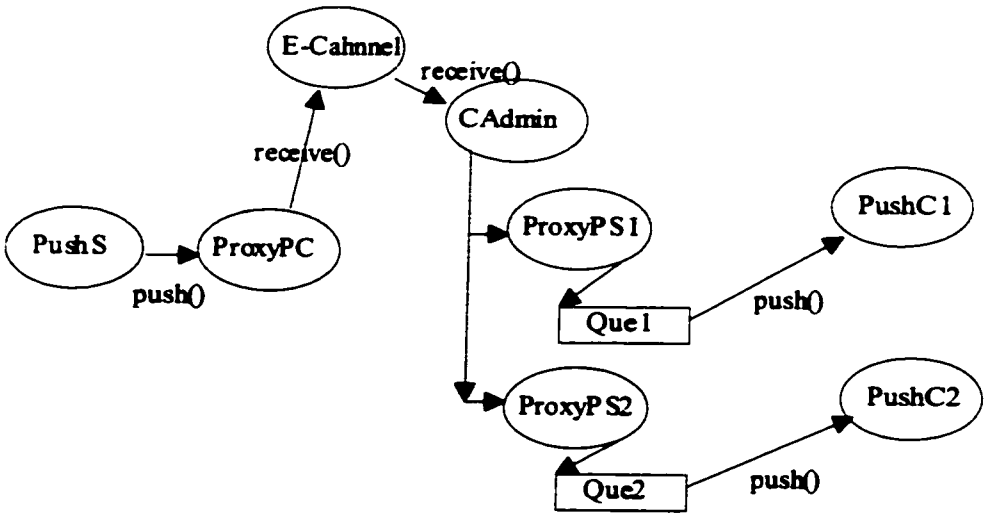

Figure 3.5 A diagram illustrate ORB3.1.3 implementation, one PushSupplier and two Push Consumer case. The PushS, ProxyPC, ProxyPS, PuchC, and Cadmin represent the objects of PushSuplier, ProxyPushConsumer, ProxyPushSupplier, PushConsumer and ConsumerAdminstration respetively.

One can certainly recognize the significant advantage of this implementation. Although suppliers still need to wait the ConsumerAdmin to finish the receive() for all of the ProxyPushSuppliers, they are all just involved with the local calls. It may be thought that an individual event queue for each ProxySupplier is an overkilled feature. Despite the extra memory and processing power incurred, this multi-queue arrangement may enhance the concurrency in the consumers' side as well. Each ProxyPushSupplier can push the event to its corresponding PushConsumer with its own designated pace. All consumers will not interfere each other. In the certain circumstance where the network traffic is highly unbalanced, the consumers with slow connection may suffer from the event dropping. However the individual problem will not cause problem in a whole, which is a desirable attribute for the distributed environment. If in the one-queue implementation, the slowest connection will dictate the speed on a whole! Some persistent storage policies may resolve the issue regarding the event dropping thus achieving guaranteed delivery

[IONA98]. Additionally, such a multi-queue arrangement is a uniform implementation when PullConsumers are present. PullConsumers must have their own event queue.
There are rooms for further improving the suppliers' side concurrency. A common event queue may be implemented at EvenChannel stage. Suppliers push calls will return after successfully inserting their events in this queue without further propagating to the ConsumerAdmin. This stems from the fact that ConsumerAdmin has to manage multiple push to all of the ProxySupplies, which could be costly. Because these calls have been local since ORBacus3.1.3, the improvement may not be so significant as compared with that from ORBacus3.1.1 to 3.1.3. Anyhow this one-queue arrangement seems to be a natural choice at the first place if no multi-queue arrangement present. We will be modeling its behavior in the next chapter.

3.6 The Properties regarding Fairness, Starvation and Logical Order

These implementations (including both V3.1.1 and V3.1.3) do not ensure fairness. There is a synchronized call at ConsumerAdmin.receive(). There could be many suppliers which may inside EventChannel.recive() call and wait for entering this synchronized call. Because the Java thread model is not a FIFO sequence, the competing ProxyPushComsumers, corresponding to suppliers, will be getting a chance to run the synchronized ConsumerAdmin.receive() in a random order. This implementation is not fair, because the propagation of the events is not based on the order of the arrival sequence of the event. Starvation could potentially occur in case of existing many pending events.
The fair and starvation free implementation could be achieved for example by employing the so-called bakery algorithm [BenAr90]. However such implementation may incur expensive overhead. In our targeted system, we are not concerned much about the arrival orders in the Event Channel. The situation for the starvation to happen will also be extremely rare. If the starvation is becoming a problem for our particular use, we could simply replace this Event Service with other more adequate one for instance the one with the characteristics discussed above. Because we are using the standard Event Service,

such replacement will be truly plug&play. Virtually no coding will be required least to say any effort related to re-engineering, which also underscores the beauty of component-based development.

On the other hand, logical order will be retained in this implementation. For example, the reply to the message M will appear after the message M at all of the consumers. Because of the synchronized ConsumerAdmin.receive(), any propagation of an event to all ProxySuplliers will be completed before starting another propagation. This will guarantee the event queue managed by each ProxySuplliers being consistent. This order will be retained by the clients when ProxyPushSupplier pushes the events. There are subtle variations for drawing collaboration regarding the consistency, which will be discussed at the design and implementation part in chapter 5.

ORBacus3.2 is also becoming available at the time of this writing. There are virtually no changes having been made pertaining to the respects discussed above. However there are two more modules being added, i.e. OBEventChannelFactory and OBTypedEventChannelFactory. These Event Channel Factories may facilitate dynamic event channels creation. They are important components when we are dealing with the conceived "Dynamic System Configuration".

Moreover, IONA's OrbixTalk [IONA98] provides a reliable and guaranteed delivery messaging system compatible with CORBA Event Service. Its MessageStore, a persistent event storage, ensures that if registered listeners miss messages through system or network difficulties, any missed messages will be forwarded as soon as the target recipient comes back online. Furthermore it employs IP multicasting protocol to save on network bandwidth and accelerate notifications. OrbixTalk is suitable for those where such reliable and guaranteed QoS is critical. There may be a performance trade off if such reliability is implemented. Due to the fact that no source code is available, we can not carry out a more detail investigation as we did for ORBacus.

4. Mathematical Modeling of the Distributed System

As we discussed in the last chapter, the different implementation of the COSEvent service (although being conform to the same interfaces) will have significant impacts on the quality of service and application logic. Sometimes even subtle variations among implementations will result in differences in various aspects from the performance to the degree of concurrency. In the severe case, a certain implementation may not be suitable to fulfill the application logic at all. Therefore, it is very desirable to have a rigorous mathematical model capable of describing the system precisely. We will develop our model using CCS, Calculus of Communicating Systems as our process notation. A variety of analysis questions related to the model with CCS can be readily answered with the tool the Concurrency Workbench (CWB). We will model an online communication system for distance education, which utilizes Event Channels as its mediator. Generally speaking, such a mathematical model or formal specification will enable the software engineer and designers to reason about the behavior of the system through simulation even before the system is built. It can also be very helpful or even crucial to validate products at the testing phase.

4.1 Essential Components and their Interactions in the System

The essential components of the distance education communication system are depicted in Figure 4.1. These components are *AU*—Authentication, *SA*—System Administrator, *S*—Student, *I*—Instructor, *EC*—Event Channel and *DB*—Database. The *DES* represents the whole system—Distance Education System.

Generally speaking, the *DES* comprises of a set of students, $S=\{S_1, S_2, \ldots, S_n\}$, a set of instructors, $I=\{I_1, I_2, \ldots, I_m\}$, a set of event channels, $EC=\{EC_1, EC_2, \ldots, EC_p\}$, one authentication component *AU*, one database instance *DB*, and one system administrator, *SA*. In principle, there could be a number of instances of *AU* as well as a number of instances of *DB*, which corresponds to the case of distributed database. For the sake of

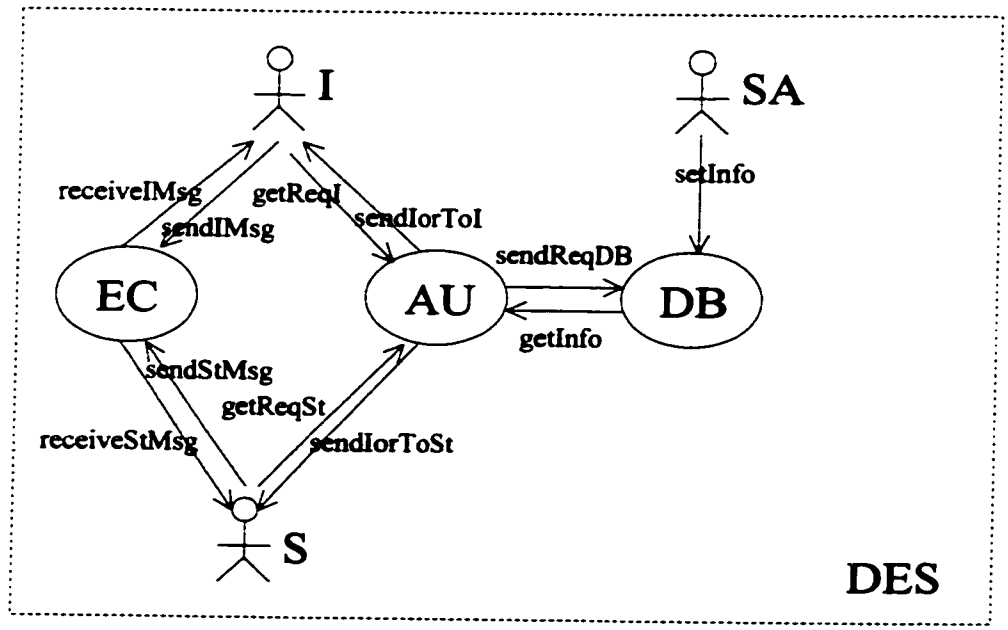

Figure 4.1 Essential components of DES and their interactions

simplicity, only centralized *AU* and *DB* are specified in our modeling. Given the fact that the *AU* component is primarily dealing with the authentication process, the job load corresponding to this component may not be significant in our interested circumstance. Therefore, it is sufficient to have only one instance of *AU* and its associated *DB*. However, it is possible to extend this centralized database case to a distributed one with certain modification. It is also possible to have a number of *SA*. For brevity, we also assume that there is only one *SA*. The components $S=\{S_1, S_2, \ldots, S_n\}$, $I=\{I_1, I_2, \ldots, I_m\}$, $EC=\{EC_1, EC_2, \ldots, EC_p\}$, *AU, DB and SA* are distributed over the heterogeneous network environment, running on different machines (or on different processes within the same machine).

The interactions or the communications between the components can be modeled as a set of actions, *Act={getReqSt, getReqI, sendReqDB, getInfo, sendIorToSt, sendIorToI, sendStMsg, receiveStMsg, sendIMsg, receiveIMsg., etc.}* as illustrated in Figure 4.1. There exist some other actions such as the connect/disconnect to the event channels etc. We will include these actions in our modeling process later on. The meaning of the actions in the *Act* set is actually pretty much self-explained. For example, the action *getReqSt,* is the interaction between *S* and *AU*, indicating that *S* issues a request for the

authentication. The action, *sendReqDB* represents the interaction between *AU* and *DB*, indicating that *AU* sends request to *DB* on behalf of the student or instructor. The action, *getInfo* represents that *AU* gets the corresponding information from the *DB* and then sends the IOR (Inter-operable Object Reference) to the student via the action *sendIorToSt*. The actions, *sendStMsg* and *receiveStMsg*, correspond to sending messages to and receiving messages from *EC*, respectively. The same interpretation applies to the case of instructor by replacing *St* (Student) with *I* (Instructor) in the above.

The functionality of *AU* is essentially to return the IOR of the desired *EC* to the authenticated users. This functionality is very similar to that of CORBA Naming Service [OMG98b]. However, CORBA Naming Service does not provide automatic authentication. In essence, when one knows the name of a CORBA object, he can use that object by obtaining its IOR from the Naming Service. In our situation, we can have more fine controls regarding the authentication by storing the information and rules in the *DB*. This could be very desirable, for instance, to the distance education system. Students can only receive the services that they are eligible to, e.g. the classes they have registered. Although such authentication information either password or the registration can be stored in a file system, *DB* could be more flexible and reliable regarding aspects of the query support and concurrency control.

4.2 Examples of Communication Patterns

Event Channel can facilitate many to many (m-to-n) communications for either push or pull at both supplier and consumer ends. In principle, the users (students or instructors) can register to be the supplier of one set of *EC*s and the consumer of another set of *EC*s. Based on the roles of the instructors and the students in the education system, we identified the following patterns as the most significant ones in the *DES*.

- Uni-direction: one to many

This case is as shown in Figure 4.2. An instructor, *I* is the supplier of the event channel *EC* and many students S_1, S_2,..., *and* S_m are the consumers of the same event channel *EC*.

This has a broadcast or multicast manner. The pattern like this corresponds to the case that the instructor is giving a "one direction" lecture to the students.

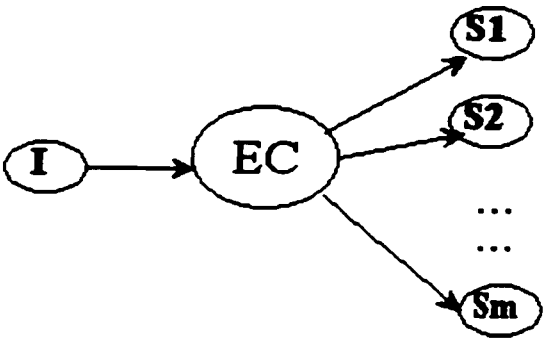

Figure 4.2 Uni-direction: one to many communication pattern

- Bi-direction, one to many and many to one

This case is shown in Figure 4.3. On one hand an instructor, *I* is the supplier of the event channel EC_1 and many students S_1, S_2,.., *and* S_m are the consumers of the same event channel EC_1. This is exactly the same to the previous case. On the other hand, S_1, S_2,..., and S_m are the suppliers to the EC_2 with the instructor *I* as its consumer. This pattern enhances the feature of the previous case by providing the feedback directly to the instructor. This arrangement will definitely increase the interactions between the instructor and the students. This is similar to the case when students can raise questions to the instructor in the classroom. However, these questions are private to the instructor and other students can not "hear" directly. Although in the traditional classroom setting, the questions will be heard on a whole. It could be beneficial to the students if the question is beating the points. But sometimes, it may interrupt the normal sequence of the lecture. With this bi-direction pattern in place, the instructor can screen the question and answer the question selectively to the whole.

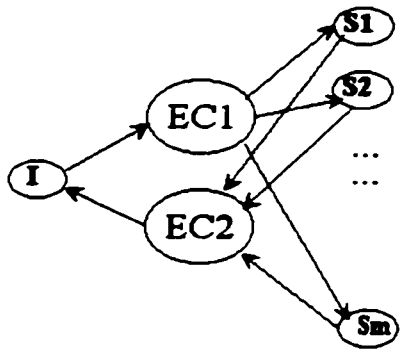

Figure 4.3 Bi-direction: one to many and many to one pattern

- Group communication: many to many (n-to-n)

Group discussion pattern provides the highest degree of interactions among the users with the least regulations. As a matter of fact, it is not necessary to make the distinction between instructors and students. They can be identified in general as users as shown in Figure 4.4. All users will register to the both ends, supplier and consumer of an Event Channel, EC. This pattern mirrors the case of tutorial or lab session in the school.

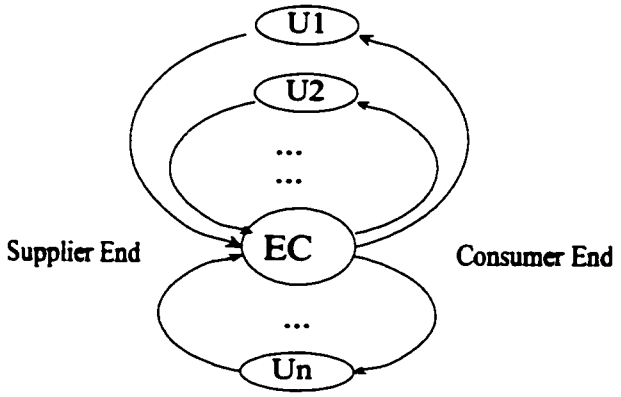

Figure 4.4 Group communication: many to many communication patterns

Although, we discuss the above communication patterns in the context of online distance education system, they could readily apply in principle to any online communication system.

4.3 CCS Notations

We will use the CCS (Calculus of Communication System) [Milne89, Bruns97] notations for our system modeling. Although other process notations (such as CSP [Hoarc85] and ACP [Bacte90]) can also be utilized, CCS turns out to be more practical and a wide variety of analysis questions can be answered with the tool, Concurrency Workbench [Bruns97].

The basic elements of CCS consist of agents, actions and operators. An agent is a part of a system whose behavior consists of discrete actions. An agent may be decomposed into sub agents acting concurrently and interactively. An action models a real world event. Each action of an agent is either an interaction with other agents through communication, or it occurs independently and concurrently. Operators control the occurrence of actions. The simple *Arbiter*, for instance, can be modeled in CCS as follows:

$$Arbiter \stackrel{def}{=} req.acq.rel.Arbiter$$

This definition states that from the state *Arbiter* a *req* (request action), then an *acq* (acquire action), and finally a *rel* (release action) transition can be performed, leading back to state *Arbiter*.

The above expression consists of the prefix "." and $\stackrel{def}{=}$ operators. We will give the precise meaning of these operators by the rules in the following.

- The prefix operator "."

The rule is: $\alpha.P \xrightarrow{\alpha} P$.

where α stands for an action and symbol P stands for a CCS expression.

The rule indicates that αP can perform an action α and thereby become P. The prefix operator is not commutative but associative.

- The definition operator $\stackrel{def}{=}$

The rule for $\stackrel{def}{=}$ is as the following:

$$\frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} A \overset{def}{=} P$$

The rule states that if an expression P can reach another expression P' by a transition α, and if A has been defined to be P, then A can also reach P' by α. Informally, one can say that if A is defined to be P then A can do whatever P can do. For brevity, we will often use "=" to replace "$\overset{def}{=}$" in our expressions.

- The summation operator "+"

The meaning of "+" is given by two rules

$$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

The first rule states that if an agent P can perform action α and becomes agent P', then $P+Q$ can also perform α and become P'. The second rule is the symmetrical case for Q. This operator models the idea of alternative or choice. $P+Q$ can do whatever P or Q can do.

Although it is often to have only binary summation such as $P+Q$, it can be applied to a general case with more than two agents ($\sum_{i \in I} P_i$) and even sum over infinitely many agents. The summation is commutative, associative and idempotent.

- The composition operator "|"

The following three rules will define the meaning of the composition:

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'} \qquad \frac{P \xrightarrow{l} P' Q \xrightarrow{\bar{l}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

The first rule says that if P can perform α and become P', the $P|Q$ can perform α and become $P'|Q$. The second rule is the symmetrical case for Q. The third rule says that if P can perform l and become P', and Q can perform $\bar{l}$ and become Q', then $P|Q$ can perform τ and become $P'|Q'$.

The actions l and $\bar{l}$ are mutual complemented, indicating that the action l can synchronize with its complemented partner $\bar{l}$. Complementation follows the rule that $\bar{\bar{l}} = l$. Therefore the synchronized action of P and Q results in a τ action because τ represents an internal activity to a system which can not be observed from outside.
The rules imply that a component of $P|Q$ either can act independently, or can synchronize with its partner to produce an internal action. Composition is both commutative and associative.

- The restriction operator "\"

The following rule will dictate the meaning of the restriction operator "\":

$$\frac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \alpha \notin L \cup \bar{L}$$

The L stands for a set of labels or actions. The rule states that $P\backslash L$ can perform any action of P except the actions in L and $\bar{L}$.

- Re-labeling

For example, if we want to define $Arbiter_1 \overset{def}{=} req_1.acq_1.rel_1.Arbiter_1$, we can use the already defined Arbiter and the re-labeling operator. That is $Arbiter_1 \overset{def}{=} Arbiter[req_1 / req, acq_1 / acq, rel_1 / rel]$. The lists of label pairs inside the square brackets define re-labeling functions. A list of the form $l_1'/l_1, \ldots l_n'/l_n$ defines functions that maps l_i to l_i' and $\bar{l}_i$ to $\bar{l}_i'$ for i from 1 to n.
The rule for the re-labeling is

$$\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f[\alpha]} P'[f]}$$

That is if P can perform α and become P', then $P[f]$ can perform $f(\alpha)$ and become $P'[f]$.
So far, we have introduced all the necessary CCS notations for our system modeling.

4.4 Modeling Distributed Education System

As discussed previously, a distance education system, *DES* comprises the following components: a set of students, $S=\{S_1, S_2, ..., S_n\}$, a set of instructors, $I=\{I_1, I_2, ..., I_m\}$, a set of event channels, $EC=\{EC_1, EC_2, ..., EC_p\}$, one authentication component *AU*, one database instance *DB*, and one system administrator, *SA*. In CCS notation, these components will be modeled as agents. These components will be running potentially on different machines concurrently and transparently. The interactions among these components are modeled as actions, which belong to the set of *Act={getReqSt, getReqI, sendReqDB, getInfo, sendIorToSt, sendIorToI, sendStMsg, receiveStMsg, sendIMsg, receiveIMsg., etc.}*. The complete collection of this *Act* set will be identified along the process of our modeling. In essence, the *DES* can be specified as the following:

$$DES = (SA|DB|AU|EC|S|I) \setminus L$$

where $S=\{S_1, S_2, ..., S_n\}$, $I=\{I_1, I_2, ..., I_m\}$, *and* $EC=\{EC_1, EC_2, ..., EC_p\}$; $L \subseteq Act$

We will further elucidate the semantics of the above expressions.

- Agent *SA*, the System Administrator:

$$SA = setInfo.SA$$

This implies that *SA* can do an action of *setInfo*—set information to *DB* and return back to the state of *SA*. *SA* is recursively defined by this way.

- Agent *DB*, the Database:

$$DB = \overline{setInfo}.DB + \overline{send\,ReqDB}.\overline{getInfo}.DB$$

The $\overline{setInfo}$ is the complement action of *setInfo*. *DB* is recursively defined. *DB* can have interactions with *SA* and return back to the *DB* after the $\overline{setInfo}$ action. On the other hand, *DB* can have interactions with *AU* (see the following paragraph for the detail). After a sequence of actions, $\overline{send\,ReqDB}.\overline{getInfo}$, *DB* returns back to its original state.

- Agent *AU*, the Authentication component

$$AU = AUS + AUI$$
$$AUS = get\,\mathrm{Re}qSt.send\,\mathrm{Re}qDB.getInfo.sendIorToSt.AU$$
$$AUI = get\,\mathrm{Re}qI.send\,\mathrm{Re}qDB.getInfo.sendIorToI.AU$$

AU is also defined recursively through the intermediate agents AUS and AUI. AUS represents the authentication process to the students. This is that AUS gets request from the student, sends this request to the DB, gets the info from DB and then returns the info (the IOR) back to the student. The same scenario applies to the AUI, corresponding to the case for the instructor. AUS and AUI have been defined in a generic way. In reality, there could be up to n students and m instructors.

We can use the relabeling method in CCS to cope with this situation.

$$AUS_i = AUS[get\,\mathrm{Re}qSt_i \,/\, get\,\mathrm{Re}qSt, sendIorToSt_i \,/\, sendIorToSt]$$
$$AUI_j = AUI[get\,\mathrm{Re}qI_j \,/\, get\,\mathrm{Re}qI, sendIorToI_j \,/\, sendIorToI]$$

Consequently, AU will be defined as:

$$AU = \sum_{i=1\ldots n} AUS_i + \sum_{j=1\ldots m} AUI_j$$

- Agent S: the set of students

Suppose that there are n students, then we have

$$S = \{S_1, S_2, \ldots, S_n\}$$

where S_i $(i=1,2,\ldots n)$ is defined as:

$$S_i = \overline{get\,\mathrm{Re}qSt_i}.\overline{sendIorToSt_i}\,connectSupStEC_{i,j}.SP_i + \overline{get\,\mathrm{Re}qSt_i}.\overline{sendIorToSt_i}\,connectConStEC_{i,k}.SC_i$$

the action $connectSupStEC_{i,j}$ represents that the student S_i is registered to the EC_j as the supplier while $connectConStEC_{i,k}$ corresponds to the case that S_i registered to the EC_k as the consumer, j, $k \in (1,2,\ldots m)$. The different subscribes j and k are used, implying that the student S_i can be the supplier and consumer of different ECs. This corresponds to the case of our example, bi-direction. If $j=k$, we have the case of group communication as discussed previously.

Further, SP_i and SC_i are defined as the following:

$$SP_i = connectConStEC_{i,k}.SPC_i + disconnectSupStEC_{i,j}.S_i + sendStMsg_{i,j}.SP_i$$
$$SC_i = connectSupStEC_{i,j}.SPC_i + disconnectConStEC_{i,k}.S_i + \overline{receiveStMsg_{i,k}}.SC_i$$
$$SPC_i = sendStMsg_{i,j}.SPC_i + \overline{receiveStMsg_{i,k}}.SPC_i + disconnectSupStEC_{i,j}.SC_i + disconnectConStEC_{i,k}.SP_i$$

The semantics of SP_i is that the student S_i has connected to a supplier end of a EC_j while SC_i indicates that S_i has connected to a consumer end of a EC_k. The semantics of SPC_i is that the student S_i has connected to a supplier end of a EC_j and a consumer end of another EC_k (it is possible that $j=k$).

S_i, SP_i, SC_i *and* SPC_i have been recursively defined. SP_i can connect to the consumer end to become SPC_i; or disconnect from the supplier end to become S_i; or send a message to the connected supplier end to return back to SP_i. The similar scenario applies to SC_i. SPC_i can send and receive messages from EC_j and EC_k receptively. SPC_i can disconnect to the supplier end to become SC_i or disconnect to the consumer end to become SP_i. The behaviors of the student have been unequivocally defined through this formal specification.

- Agent I: the set of Instructors

Suppose that there are m instructors, then we have

$$I = \{I_1, I_2, ..., I_m\}$$

where I_i $(i=1,2,...m)$ is defined as

$$I_i = \overline{get\,\mathrm{Re}qI_i}.\overline{sendIorToI_i}\,connectSupIEC_{i,j}.IP_i + \overline{get\,\mathrm{Re}qI_i}.\overline{sendIorTIt_i}\,connectConIEC_{i,k}.IC_i$$

Similar to the previous case, the action $connectSupIEC_{i,j}$ represents that the instructor I_i is registered to the EC_j as the supplier while $connectConIEC_{i,k}$ corresponds to the case that I_i registered to the EC_k as the consumer, $j, k \in (1,2,..\ m)$. The different subscribes j and k are used, implying that the I_i can be the supplier and consumer of different ECs. This corresponds to the case of our example, bi-direction. If $j=k$, we have the case of group communication as discussed previously. However if only the action $connectSupIEC_{i,j}$ is conducted, we are in the situation of one direction (one to many) pattern as shown example previously.

Furthermore, IP_i and IC_i can be defined as the following:

$$IP_i = connectConIEC_{i,k}.IPC_i + disconnectSupIEC_{i,j}.I_i + sendIMsg_{i,j}.IP_i$$

$$IC_i = connectSupIEC_{i,j}.IPC_i + disconnectConIEC_{i,k}.I_i + \overline{receiveIMsg_{i,k}}.IC_i$$

$$IPC_i = sendIMsg_{i,j}.IPC_i + \overline{receiveIMsg_{i,k}}.IPC_i + disconnectSupIEC_{i,j}.IC_i + disconnectConIEC_{i,k}.IP_i$$

The semantics of IP_i is that the instructor I_i has connected to a supplier end of a EC_j while IC_i indicates that I_i has connected to a consumer end of a EC_k. The semantics of IPC_i is that the instructor I_i has connected to a supplier end of a EC_j and a consumer end of another EC_k (it is possible that $j=k$).

I_i, IP_i, IC_i and IPC_i have been recursively defined. IP_i can connect to the consumer end to become IPC_i; or disconnect from the supplier end to become I_i; or send a message to the connected supplier end to return back to IP_i. The similar scenario applies to IC_i. IPC_i can send and receive messages from EC_j and EC_k receptively. IPC_i can disconnect to the supplier end to become IC_i or disconnect to the consumer end to become IP_i. Similar to the case of students, the behaviors of the instructor have also been unequivocally defined through this formal specification.

- Agent EC: the set of Event Channels

Suppose that there are p Event Channels, then we have

$$EC = \{EC_1, EC_2, \ldots, EC_p\}$$

where

$$EC_k = \sum_{i=1,2\ldots n} \overline{connectSupStEC_{i,k}}.EC_k + \sum_{i=1,2\ldots n} \overline{connectConStEC_{i,k}}.EC_k + \sum_{i=1,2\ldots n} \overline{disconnectSupStEC_{i,k}}.EC_k + \sum_{i=1,2\ldots n} \overline{disconnectConStEC_{i,k}}.EC_k + \sum_{i=1,2\ldots m} \overline{connectSupIEC_{i,k}}.EC_k + \sum_{i=1,2\ldots m} \overline{connectConIEC_{i,k}}.EC_k + \sum_{i=1,2\ldots m} \overline{disconnectSupIEC_{i,k}}.EC_k + \sum_{i=1,2\ldots m} \overline{disconnectConIEC_{i,k}}.EC_k + \sum_{i=1,2\ldots n} \overline{sendStMsg_{i,k}}.enQue_k.ECQ_k + \sum_{i=1,2\ldots m} \overline{sendIMsg_{i,k}}.enQue_k.ECQ_k$$

In the above expressions, apart from $enQue_k$ all actions (more precisely, their uncomplemented partners) have been defined in Agent S and Agent I. For example, $connectSupStEC_{i,k}$ has been defined in Agent S, implying that the student S_i is connecting

to the supplier end of Event Channel k. The complementation action of this is $\overline{connectSupStEC_{i,j}}$ as defined in the above. The action $enQue_k$ is related to the insertion of an element to the event queue of EC_k. The definition and semantics of these queues will be presented in the next section.

ECQ_k is the intermediate representation, which implies that EC_k has events stored in its queue and is ready to be forwarded to its consumers. ECQ_k is defined as the following:

$$ECQ_k = deQue_k.(\prod_{j_s} receiveIMsg_{j_s,k}.)(\prod_{i_t} receiveSMsg_{i_t,k}.)ECQX_k$$

$$ECQX_k = empty_k.EC_k + notEmpty_k.ECQ_k$$

where $\prod_{j_s} receiveIMsg_{j_s,k}. = receiveIMsg_{j_1,k}.recevieIMsg_{j_2,k}....receiveIMsg_{j_u,k}.$

$\prod_{i_t} receiveSMsg_{i_t,k}. = receiveSMsg_{i_1,k}.recevieSMsg_{i_2,k}....receiveSMsg_{i_v,k}.$

There are u instructors and v students registered as consumers of event channel k (obviously, $u<=m$, and $v<=n$). Once there are stored events in event channel, the event channel will remove the oldest event and forward it via *receiveMsg* to all its consumers one by one.

After its sending the event to all of its registered consumers, the event channel will enter an intermediate state defined as $ECQX_k$. This state will have synchronized actions *empty* or *notEmpty* with its queue. If the queue is empty, the event channel will return back to EC_k. Otherwise, it will return to ECQ_k. EC_k has been recursively defined.

- Agent Q: the set of Queues

Each Q_k will be associated with its event channel EC_k, we thus have

$$Q = \{Q_1, Q_2, ..., Q_p\}$$

As we discussed previously about some detail implementation of event channels, we came to the point that event queue within the EC is very crucial for the quality of service. The push-push notification communications could be implemented even without a queue, as the case of ORBacus3.1.1. This kind of configuration is hardly to provide any

decoupled communications. The supplier will be directly engaged to all consumers when pushing an event. Given the time cost for each remote method invocation, such direct engagement is highly undesirable. Therefore it is important to model the queues associated with the event channels. What we specify here is very fundamental, i.e. there exists one event-queue for each event channel. The length of the event queue is *l*, i.e. up to *l* events can be stored in the queue. If there are more than *l* events inside the queue, the oldest event will be removed to satisfy length_of_queue =*l*.

To meet certain quality of service, the specified characteristics of the queue is essential. Nevertheless, more advance approach may be utilized in the implementation, such as one queue for every consumer in a channel as in the case of ORBacus3.1.3.

The queue Q_k or $Q_k^{(l)}$ is defined as follows:

$$Q_k^{(0)} = \overline{enQue_k}.Q_k^{(1)} + \overline{empty_k}.Q_k^{(0)}$$
$$Q_k^{(1)} = \overline{enQue_k}.Q_k^{(2)} + \overline{deQue_k}.Q_k^{(0)} + \overline{notEmpty_k}.Q_k^{(1)}$$
$$\ldots$$
$$Q_k^{(i)} = \overline{enQue_k}.Q_k^{(i+1)} + \overline{deQue_k}.Q_k^{(i-1)} + \overline{notEmpty_k}.Q_k^{(i)}$$
$$\ldots$$
$$Q_k^{(l)} = \overline{enQue_k}.\overline{deQue_k}.Q_k^{(l)} + \overline{deQue_k}.Q_k^{(l-1)} + \overline{notEmpty_k}.Q_k^{(l)}$$

where $Q_k^{(i)}$ will be turned into $Q_k^{(i+1)}$ and $Q_k^{(i-1)}$ respectively following the complemented actions of $enQue_k$ and $deQue_k$. Obviously, $Q_k^{(0)}$ can not have an action of $deQue_k$. The semantics of $\overline{enQue_k}.\overline{deQue_k}.Q_k^{(l)}$ is that the oldest event will be removed upon the insertion of the $(l+1)$ element. Clearly only $Q_k^{(0)}$ will be synchronized with $empty_k$ and the rest with $notEmpty_k$.

Up to this point, we have modeled all of the essential components for our distributed education system including the event queues. Based on this model, a formal specification can be generated when the system parameters are given. These parameters include the number of event channels (p), the number of the students (n), the number of the instructors (m), the length of the queue (l), and the communication pattern for the particular channels. The behavior of the system could be tested and reasoned with the tool, such as Concurrency Workbench (CWB) [Bruns97]. The whole system can be further implemented based on this specification. By employing the distributed object

technology such as CORBA, the interfaces of the components can be further specified with CORBA IDL. This enables the components to be separately developed or purchased. These components can be deployed in different machine nodes over the Internet. With CORBA as its component middleware or infrastructure, these components will be running concurrently on each machine and integrated seamlessly in a whole over the heterogeneous network.

4.5 Summary of the System Modeling and an Example Illustration

In the last section, we have presented our mathematical models in a step by step fashion and discussed the semantics pertaining to them. In addition to the agents of *SA, DB, AU, S, I,* and *EC,* we also modeled the queue *Q* associated with *EC.* Although *Q* is logically internal to *EC,* we have to explicitly include *Q* in the system *DES* since there are actions such as *enQue, deQue, empty,* and *notEmpty* associated with it. However other agents such as *SP, SPC, ECQ* etc. do not need to be included explicitly. This is because they are just intermediate steps in defining *S, EC* recursively.

Therefore we finally model the distance education system as:

$$DES = (SA|DB|AU|EC|Q|S|I) \setminus L$$

where $S=\{S_1, S_2, \ldots, S_n\}$, $I=\{I_1, I_2, \ldots, I_m\}$, $EC=\{EC_1, EC_2, \ldots, EC_p\}$ *and* $Q=\{Q_1, Q_2, \ldots Q_p\}$;

$L = \{setInfo, send\,\mathrm{Req}DB, getInfo\} \cup Get\,\mathrm{Req} \cup SendIorTo \cup SendMsg \cup \mathrm{Re}ceiveMsg$

$\cup\ ConnectEC \cup DisconnectEC \cup EnQue \cup DeQue \cup Empty \cup NotEmpty$

and

$$Get\,\mathrm{Req} = \bigcup_{i=1\ldots n} \{get\,\mathrm{Req}St_i\} \bigcup_{j=1\ldots m} \{get\,\mathrm{Req}I_j\}$$

$$SendIorTo = \bigcup_{i=1\ldots n} \{sendIorSt_i\} \bigcup_{j=1\ldots m} \{sendIorI_j\}$$

$$SendMsg = \bigcup_{i=1\ldots n}\bigcup_{s=1\ldots p} \{sendStMsg_{i,s}\} \bigcup_{j=1\ldots m}\bigcup_{t=1\ldots p} \{sendIMsg_{j,t}\}$$

$$\mathrm{Re}ceiveMsg = \bigcup_{i=1\ldots n}\bigcup_{s=1\ldots p} \{receiveStMsg_{i,s}\} \bigcup_{j=1\ldots m}\bigcup_{t=1\ldots p} \{receiveIMsg_{j,t}\}$$

$$ConnectEC = \bigcup_{i=1\ldots n}\bigcup_{s=1\ldots p} \{connectSupStEC_{i,s}\} \bigcup_{j=1\ldots m}\bigcup_{t=1\ldots p} \{connectSupIEC_{j,t}\}$$

$$\bigcup_{i=1\ldots n}\bigcup_{s=1\ldots p} \{connectConStEC_{i,s}\} \bigcup_{j=1\ldots m}\bigcup_{t=1\ldots p} \{connectConIEC_{j,t}\}$$

$$DisconnectEC = \bigcup_{i=1\ldots n}\bigcup_{s=1\ldots p} \{disconnectSupStEC_{i,s}\} \bigcup_{j=1\ldots m}\bigcup_{t=1\ldots p} \{didconnectSupIEC_{j,t}\}$$

$$\bigcup_{i=1\ldots n}\bigcup_{s=1\ldots p} \{disconnectConStEC_{i,s}\} \bigcup_{j=1\ldots m}\bigcup_{t=1\ldots p} \{disconnectConIEC_{j,t}\}$$

$$EnQue = \bigcup_{i=1..p} \{enQue_i\}$$
$$DeQue = \bigcup_{i=1..p} \{deQue_i\}$$
$$Empty = \bigcup_{i=1..p} \{empty_i\}$$
$$NotEmpty = \bigcup_{i=1..p} \{notEmpty_i\}$$

The semantics of all above actions can be found in the last section, which also includes definitions of all agents.

As an example, let's assume a *DES* system with one instructor, two students, and one *EC* channel with uni-direction one-to-many communication pattern and event queue length of 2. That is the case where $n=2, m=1$ and $p=1$ and $l=2$. The whole system can be specified as the following:

$$DES = (SA|DB|AU|EC|Q|S|I) \setminus L$$

where $S=\{S_1, S_2\}$, $I=\{I_1\}$, $EC=\{EC_1\}$ and $Q=\{Q_1\}$

and $L=\{setInfo, sendReqDB, getInfo, getReqSt_1, getReqSt_2, getReqI_1, sendIorToSt_1, sendIorToSt_2, sendIorToI_1, sendIMsg_{1.1}, receiveStMsg_{1.1}, receiveStMsg_{2.1}, connectSupIEC_{1.1}, connectConStEC_{1.1}, conectConStEC_{2.1}, disconnectSupIEC_{1.1}, disconnectConStEC_{1.1}, disconectConStEC_{2.1}, enQue_1, deQue_1, empty_1, notEmpty_1\}$.

The semantics of the agents are as the following,

$$SA = setInfo.SA$$

$$DB = \overline{setInfo}.DB + \overline{send\ \mathrm{Re}qDB}.\overline{getInfo}.DB$$

$$AU = get\ \mathrm{Re}qSt_1.send\ \mathrm{Re}qDB.getInfo.sendIorToSt_1.AU + get\ \mathrm{Re}qSt_2.send\ \mathrm{Re}qDB.getInfo.sendIorToSt_2.AU + get\ \mathrm{Re}qI_1.send\ \mathrm{Re}qDB.getInfo.sendIorToI_1.AU$$

$$S_1 = \overline{get\ \mathrm{Re}qSt_1}.\overline{sendIorToSt_1}\,connectConStEC_{1.1}.SC_1$$

$$SC_1 = disconnectConStEC_{1.1}.S_1 + receiveStMsg_{1.1}.SC_1$$

$$S_2 = \overline{get\ \mathrm{Re}qSt_2}.\overline{sendIorToSt_2}\,connectConStEC_{2.1}.SC_2$$

$$SC_2 = disconnectConStEC_{2.1}.S_2 + receiveStMsg_{2.1}.SC_2$$

$$I_1 = \overline{get\ \mathrm{Re}qI_1}.\overline{sendIorToI_1}\,connectSupIEC_{1.1}.IP_1$$

$$IP_1 = disconnectSupIEC_{1.1}.I_1 + sendIMsg_{1.1}.IP_1$$

$$\begin{aligned} EC_1 = {} & \overline{connectConStEC_{1.1}}.EC_1 + \overline{connectConStEC_{2.1}}.EC_1 \\ & + \overline{disconnectConStEC_{1.1}}.EC_1 + \overline{disconnectConStEC_{2.1}}.EC_1 \\ & + \overline{connectSupIEC_{1.1}}.EC_1 + \overline{disconnectSupIEC_{1.1}}.EC_1 \\ & + \overline{sendIMsg_{1.1}}.enQue_1.ECQ_1 \end{aligned}$$

$$ECQ_1 = deQue_1.receiveSMsg_{1.1}.receiveSMsg_{2.1}.ECQX_1$$

$$ECQX_1 = empty_1.EC_1 + notEmpty_1.ECQ_1$$

$$Q_1^{(0)} = \overline{enQue_1}.Q_1^{(1)} + \overline{empty_1}.Q_1^{(0)}$$

$$Q_1^{(1)} = \overline{enQue_1}.Q_1^{(2)} + \overline{deQue_1}.Q_1^{(0)} + \overline{notEmpty_1}.Q_1^{(1)}$$

$$Q_1^{(2)} = \overline{enQue_1}.\overline{deQue_1}.Q_1^{(2)} + \overline{deQue_1}.Q_1^{(1)} + \overline{notEmpty_1}.Q_1^{(2)}$$

The above semantic expressions can be directly mapped into the CWB agents expressions. Software engineers are thus able to explore the system behaviors with the simulator or other facilities available with CWB. For more complicated situation, the process to convert the specification to CWB agents expressions for given parameters could be very tedious. This conversion process can be made automatically by implementing a software tool. Following the modeling and specification presented thus far, it should be straightforward to implement such a tool. However the implementation of this conversion tool is beyond the scope of this thesis.

Moreover at a recent work [Tjand99], assembly line pattern with the distributed system has been investigated. Only the so-called linear fashion was addressed, i.e. there exist exact a pair of one supplier and one consumer for each channel. In that sense, our model has addressed certain non-linear features in the communication system.

To summarize, we have presented a rigorous mathematical modeling of the object-based, real-time communication system for distance education. This model can be used to generate the formal specification, which allows software engineers to examine the behavior of the system even before the starting of the implementation. The significant parts of this modeling include follows: 1) Essential components in distance education communication system have been identified. 2) The interactions among the components have been identified. 3) Three most significant communication patterns have been

classified. 4) Rigorous mathematical modeling and formal specification have been provided. All of the above constitutes an array of contributions of this thesis. This is the first time that such mathematical modeling has been applied to the distance education system.

5. System Design and Implementation

The system design will be primarily based on the formal specification presented in the last chapter. Although the behavior of the system has been rigorously specified, there are still many issues to be addressed in the design and implementation. Many design decisions and trade-off will be described as we proceed the discussion. We will follow the methodology of component-based design. Because our targeted audiences are Internet users, we choose Java as the primary implementation language due to its high portability. The middleware is built with CORBA compliant product, ORBacus [OOC99a]. The result of this implementation will be a prototype of Object-Based, Distributed Online Real-time Communication System. This prototype will be served as a proof of concepts of such distributed communication system formally specified at chapter 4, which will in turn validate that the proposed approach can be successfully applied to our domain. This system shall be able to handle real-time communication for a range of media in a uniform manner. Moreover this prototype will have sufficient functionalities and capacities to be useful in real world.

In this chapter, we will start with the description of the development environment. We then discuss the rational of design and present the system architecture, followed by the detailed design of the system, IDL interfaces and description of the implementation. Finally, we will describe the operation of the system, enlist requirements for such a system to operate and discuss rooms for improvement.

5.1 Development Environment

5.1.1 Platform

The implementation of the Distributed Communication System is primarily carried out on an NT4.0 Workstation. This machine is a Gateway 2000 with a Pentium 233Mhz processor and 96MB RAM. Since the essential components of this system are implemented with Java, the platform is not really an issue here due to the fact of Java's WORA (Write Once Run Anywhere). This is especially true for the user side, which is

supposed to be deployed to the heterogeneous Internet environment where diverse platforms are expected. The IP address of this machine is 137.207.16.20 with a domain name of 3139erie-13.lams.uwindsor.ca (it used to be named as Arjuna).

Additionally, in order to facilitate the video/audio captures and playing, a sound card and video capture accessory are required. The sound card is provided originally with the Gateway PC, which is a Sound Bluster PCI. The video camera is Logitech QuickCam VC. The major specifications for this video capture device are as follows: 1) capture up to 24-bit colors; 2) resolution with 352x288 pixels; 3) capture up to 30 fps (frames per second) based on image size, resolution and user's system. The camera is connected with the PC through the parallel port, in which case no capture card is needed. This type of camera is sufficient for the current development. The parallel input setup of this NT is set as DMA (Direct Memory Access) mode, which will enhance the capture performance. If even better performance is required for video capture, the PCI type capture card can be installed. This card can be connected with standard video camera (or VCR as input) for better quality as well.

The NT Workstation turns out to be the right choice for this implementation because of the following: the easy of use, availability of software and hardware, cost, and sophistication of operation system model.

5.1.2 Development Tools

As stated previously, the implementation language will be the Java language primarily due to its unrivaled portability. Additional reasons to choose this language include follows: 1) Java is a pure Object Oriented Language (OOL). OOL is believed to be the most suitable language for component-based development [Brown98]. 2) Java has some unique and nice features such as automatic garbage collection, exception handling, Internet ready, integrated thread support. 3) Its rich portable APIs and many free available tools make it an ideal tool for our system development.

The middleware or distributed infrastructure will be CORBA. This is due to the following facts: 1) CORBA is one of industry standard for middleware and component infrastructure, endorsed by 800+ companies. 2) CORBA is suitable for heterogeneous

environment [Meta98]. 3) Tools are available for Java CORBA mapping. 4) Standard object services are available.

In the following, we will list an array of development tools that have been used in the system development. However they are not equally important or used with similar frequency in the whole development process. Core Java and ORBacus/CORBA are the major tools, while the rest ones are used at certain part of the development.

- **JDK1.2:** JDK1.2 is the latest Java Version for Java 2 platform at the time of this development. It has much more powerful APIs class libraries than those of JDK1.1. In particular the Java Foundation Classes (JFC or Swing) for GUI is becoming a standard core part for this release. Visual J++ is also used in the development. Because of its incompatibility with JDK1.2, we just use it as an editing and organizing tool instead of an IDE (Integrated Development Environment).
- **ORBacus** (CORBA middleware tool) with its Basic Object Services such as Naming Service, Event Service, etc. Together with this tool is its Java to IDL compiler—jidl, which will translate the CORBA IDL to Java code to be used in connection with ORBacus. ORBacus3.2 is JDK1.1 compatible, however it will work with Java 2 platform pretty smoothly with a bit of getting around i.e. setting correct system properties, fixing a few deprecated methods etc.
- **Visual C++ and Java Native Interface, JNI:** We will use native implementation (Visual C++) for video capture. Visual C++ is used to compile the necessary DLL files and Java JNI is used to integrate C++ part to the Java environment.
- **Java Media Framework JMF2.0**: This is a standard Java extension, which provides a Java solution for cross platform media playing and video/audio capture. In this work, JMF is used primarily to facilitate video/audio playing. Due to performance consideration, we use native implementation (C++) for video capture.
- **DBMS:** Oracle8i Enterprise Edition has been installed in this NT Workstation to serve as a database server. Various information or data can be stored to and retrieved from this database. We can leverage its built-in SQL support, concurrency control, security etc. for more robust application in compare to the flat file implementation.

For example, authentication component could use DBMS for storing client information. All other persistent data and state can be managed by the DBMS as well.

- **JDBC:** Database access is through JDBC, which is part of Java 2 platform. In this work JDBC is used to access Oracle8i database through CORBA middleware or Java Servlet in connection with Web Server.
- **Web Server and Servlet:** Apache 1.3.6 is installed in the NT workstation. Therefore our distributed objects system can be closely interacted with web server. The web server can host document and retrieve information from the database through 3-tier architecture with Java Servlet/JDBC.

5.2 Design Considerations and System Architecture

5.2.1 Some Design Considerations

There are various ways in building a multi-party communication system over a network. It could be an approach using TCP/IP (UDP/IP) socket with client/server. It could be an approach with distributed object utilizing standard Object Services. Our primary concern is also to have a uniform communication mechanism for various media types.

5.2.1.1 Deficiency of the Conventional Client/Server Approach

Let's think about a scenario for a typical client/server model for the group communication. Even for the simplest character-based client/server chat program, many careful considerations are still involved for writing robust applications. First the communication protocol must be "invented". This protocol has to be decent enough in the first place. Any modification of this protocol may be expensive. As an example for Java socket programming, the simplest approach may use readLine()/println() in the Input/Output streams, which requires no parsing. Then the server must be implemented as multi-threaded one. To be more specific, each client may be assigned as one corresponding thread. In addition, there will be a writer thread, which will write the received message to every client. All the threads must be synchronized in some way in order not to be messed up with each input. The thread for this writer may have higher

priority than other threads, depending on the policy of the server. The wait()/notify() could be implemented in order to reduce the unnecessary resource consumption. The input from each client could be placed in a queue for the writer to process. In this way, client input could be more responsive—no need to wait the writer processing, and fairness could be ensure by implementing FIFO waiting queue.

As one can see from the above, it requires many considerations for an even very simple chat program design. However, there are several problems involved in this design. First of all, the application is not well modularized, where the communication part is tightly integrated with the particular application. Second, it is not easy to extend. If we need kind of parsing, the protocols have to be redesigned. If serialized objects or images have to be incorporated, complete redesign could be the best solution. Third, how to handle the scalability? One possible approach could be the duplication of the servers. The server to server communication protocol will be required. The development process is not uniform at all. Furthermore the while loop for each client thread will consume the CPU resource.

5.2.1.2 Distributed Object Solution and Using CORBA Event Service as a Central Hub

Based on the discussions in the last section, we will employ CORBA Event Service as a central hub for our communication system. The major benefits of this approach are as follows:

1) The event service is CORBA standard basic service. The component-based development is to integrate the existing component instead of building form scratch. Therefore this approach is conform to the principle of CBD.
2) The generic event service, which can propagate all type—ANY type of the event regardless whether it is string, image or drawings. CORBA IDL provides a standard way to encode and decode those types to and from ANY. Therefore no adhoc or self-invented protocol required for each of event types. This shall provide a uniform solution for a class of communications.
3) Maintenance issues: because of well-defined interfaces, each module can be maintained and upgraded separately. For example, event service module can be

replaced with ones from different vendors based on requirements of performance or Quality of Services.

4) Flexibility in composing of the Event Channels. The output of one channel could be the input of the others. So channels can be freely composed in a sequential or parallel way. The communication between the channels is exactly the same with that between client and channels. This enables the load to be transferred from one channel to the other.
5) Scalability: Event Channels are distributed over the different machines. The system can have high scalability.
6) Standard CORBA Objects, which can leverage the security of IOR and many other layers.

5.2.1.3 Authentication Management

We need to consider issues regarding authorization/authentication and securities, and the way to provide IOR (Interoperable Object reference) to users. A standard way to make IOR available to the users is through the standard CORBA Naming Service. However this approach alone does not lend us an ability to handle authentication process. Although commercial available SSL will provide a graceful way to handle security issues, the service itself is very expensive and may not be tailored to our particular need.

In addition to the Naming Service, we design a CORBA middleware component named Coordinator that will handle the IOR registration and retrieval. This component further employs the JDBC to connect the Oracle server where the data and user information are stored. Although we can use the flat file system to store the information, the benefit of employing a DBMS is evident. For example, we can leave the programmer's responsibility for concurrency and security control to DBMS. Additionally DBMS has SQL support as a standard. The optimization of the queries is automatically performed by the DBMS engine. This will alleviate our concerns in devising efficient means in searching information in many different files. Furthermore we automatically have a highly scalable information storage system. Therefore in this system DBMS is used to assist our system management in addition to a system repository.

5.2.2 System Architecture

Figure 5.1 is an illustration of an overall system architecture. This diagram consists of some similar components to the model component view in Figure 4.1. However, the current diagram exhibits a more concrete view, which could be a real deployment. The architecture shows an example of our distributed communication system in distance education.

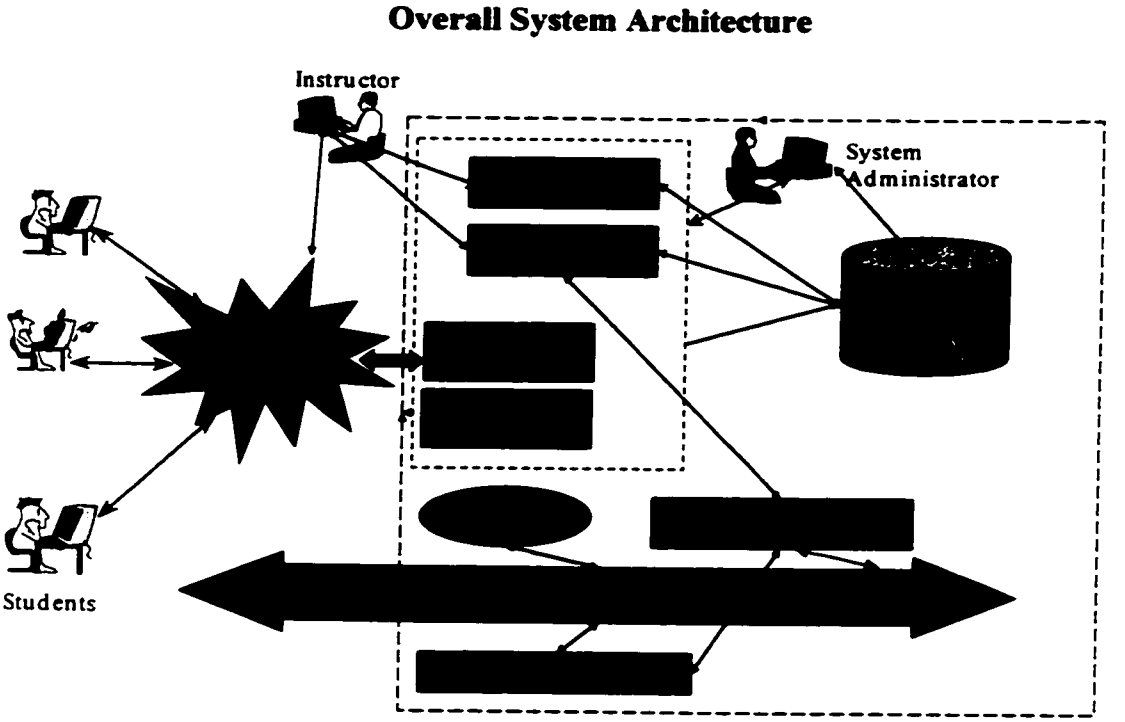

Figure5.1 An illustration of overall system architecture

5.2.3 A Typical Graphical User Interface

Figure 5.2 shows a typical GUI for the online real-time communication and collaboration system. We put this illustrative representation at this stage in order to make our discussions in the following sections easier. In essence, there are four panel areas within the main window frame. The upper-left panel is the area for drawing, the upper-right is

the area for image presentation, the middle panel is for the control of media playing, and the bottom panel is the place for chatting. A click of the "Connect" button will connect the user to the desired Event Service. The subsequent real-time communication and collaboration will be possible to conduct. The functionality of many buttons in the figure is pretty much self-explained. The menu bar is the place containing many menu items to trigger different functionality. The overall GUI component is named as CyberClassRoom since it facilitates classroom-like or tutorial-like experience in Cyber space. The GUI is implemented with Java Swing.

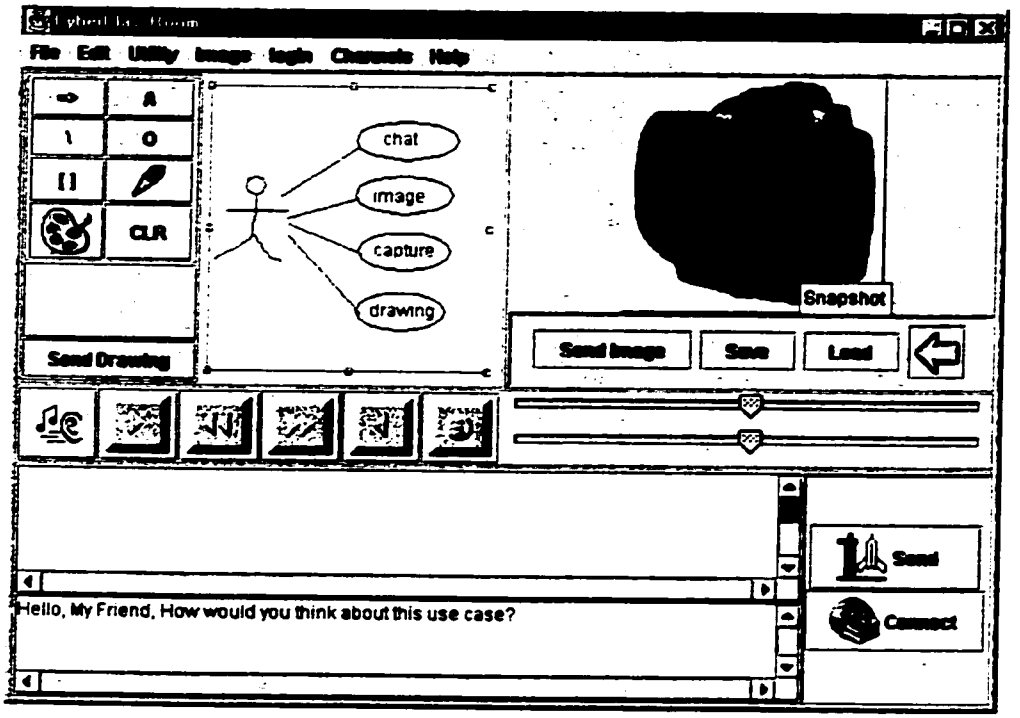

Figure 5.2 A typical GUI for online real-time collaborations.

5.3 The Contracts among Components—IDL Interfaces

In the paradigm of component-based development, interfaces are the contracts between the service providers and the service consumers. When CORBA is used as the middleware for distributed object development, interfaces are prescribed with CORBA IDL. IDL interfaces are grouped into modules. In this section IDLs for a set of modules being used in the system will be presented.

5.3.1 The IDL for Event Supplier and Consumer--CosEventComm

The module CosEventComm (stands for CORBA Object Service Event Communication) consists of interfaces for Push Consumer, Push Supplier, Pull Consumer and Pull Supplier. These interfaces have been defined by OMG at its CORBA Basic Service Specification [OMG98b].

```
module CosEventComm{

        exception Disconnected {};

        interface PushConsumer{
                void push(in any data) raises(Disconnected);
                void disconnect_push_consumer();
        };

        interface PushSupplier{
                void disconnect_push_supplier();
        };

        interface PullConsumer{
                void disconnect_pull_consumer();
        };

        interface PullSupplier{
                any pull() raises(Disconnected);
                any try_pull(out boolean has_event) raises(Disconnected);
                void disconnect_pull_supplier();
        };
};
```

5.3.2 The IDL for Event Administration--COSEventAdmin

The module CosEventChannelAdmin (stands for CORBA Object Service Event Channel Administration) consists of interfaces for Event Channel, Supplier and Consumer Administrations, and Proxies for Push/Pull Consumer/Supplier. The interfaces of this module have been defined by OMG at its CORBA Basic Service Specification [OMG98b].

```
#include <CosEventComm.idl>

module CosEventChannelAdmin{

        exception AlreadyConnected {};
```

```
        exception TypeError {};

        interface ProxyPushConsumer : CosEventComm::PushConsumer{
                void connect_push_supplier(in CosEventComm::PushSupplier push_supplier)
                raises(AlreadyConnected);
        };

        interface ProxyPullSupplier : CosEventComm::PullSupplier{
                void connect_pull_consumer(in CosEventComm::PullConsumer pull_consumer)
                raises(AlreadyConnected);
        };

        interface ProxyPullConsumer : CosEventComm::PullConsumer{
                void connect_pull_supplier(in CosEventComm::PullSupplier pull_supplier)
                raises(AlreadyConnected, TypeError);
        };

        interface ProxyPushSupplier : CosEventComm::PushSupplier{
                void connect_push_consumer(in CosEventComm::PushConsumer push_consumer)
                raises(AlreadyConnected, TypeError);
        };

        interface ConsumerAdmin{
                ProxyPushSupplier obtain_push_supplier();
                ProxyPullSupplier obtain_pull_supplier();
        };

        interface SupplierAdmin{
                ProxyPushConsumer obtain_push_consumer();
                ProxyPullConsumer obtain_pull_consumer();
        };

        interface EventChannel{
                ConsumerAdmin for_consumers();
                SupplierAdmin for_suppliers();
                void destroy();
        };
};
```

5.3.3 The IDL for Message Passing and Coordinator

IDL for Messages (passing through Event Service) is defined in the module of jidl as shown in the following. Generic (or untyped) Event Channel is used in this system, where the transferred event messages are the type of CORBA any. However the message must be an IDL type. Therefore the content of the module jidl is basically the IDL type definition. The chatting message is a string type that is a primitive IDL type, thus needs no further definition. Therefore the following IDL type definitions are essentially for passing drawing objects.

```
//IDL
//Drawing.idl
//IDL type for all of the drawing Object
module jidl{

        struct PointCORBA{
                long x;
                long y;
        };

        //represent the FreeHandDrawing
        typedef sequence <PointCORBA> PointSequence;

        struct DrawCORBA{
                string hostID;
                string timeID;
                long color;
                long shapeCode; //Rect, Oval, Line, ect.

                long x; //Rectangle rect representation;
                long y;
                long w;
                long h;


                //other parameters
                long lineCode;
                string content;
                PointSequence points;
        };

        struct UpdateCORBA{
                //Remote Update is limited to the variation
                //of the size and the location of the drawing object
                string hostID;
                string timeID;
                long x;//Rectangle rect represntation;
                long y;
                long w;
                long h;
        };
};
```

The IDL type, DrawCORBA represents a structure that corresponds to drawing objects of a line, a rectangle, an oval, a text content or a freehand drawing. Any drawing object shall be associated with a bound rectangle corresponding to the fields of x, y, w and h. A drawing object can be updated and its effect shall be reflected to drawings of all the users. This is the purpose of the IDL type of UpdateCORBA. The drawing update is limited to the changes of size and location, thus only the rectangle bound information is being sent. The combination of hostID and timeID (the time of which a drawing object is created at

the user machine) can be used to uniquely identify a drawing object. The IDL type DrawCORBA includes a wide range of basic shapes. If additional drawing types need to be added, new types- could be added to this module.

The following simple IDL type definition is singled out from the above module because it is exclusively dealing with the image type passing. Image passing is devised in using a byte array type. In fact it is a "sequence" of "octet" in IDL, which allows the length of array to be determined at run time. Although being simple, it turns out to be a very effective way in passing images. In principle any image in the memory can be serialized to a byte array as the way to store to a file. This byte array is being defined now as an IDL type, thus is able to be sent via Event Channel to all parties. This byte array can be reconstructed to the image as the way being loaded from a file. More discussion about image message passing will be presented in the section that is dealing with detailed design.

```
//IDL
//ByteArray.idl
module jidl{
  typedef sequence <octet> ByteArray;
};
```

Generally speaking, drawing objects can also be sent via the byte array type. At the point of sending, the drawing object can be serialized to a byte array with the help of Java Serialization Framework. At the point of receiving, the drawing object can be reconstructed through the deserialization process. This may be served as an alternative approach for future modification. Although being very simple, the serialization and deserilization process may incur additional overhead. Nevertheless, the current approach provides a good demo that three different IDL types i.e. built-in string, composite sequence and user defined structure type, can be effectively and uniformly transferred with generic Event Channels.

The IDL interface defined in module Coordinator corresponds to the component in handling the authentication and management. There are several exceptions defined as well. This component is functioning as a coordinator as its name indicated. It is connected

with DBMS on one hand and provides users (Students and Instructors) with authentication services on the other.

```
//IDL File
//Coordinator.idl
module coordinator{
        //throws exception when there is no such user
        //or the passwd and login name not match
        exception NotAuthorizedException{string msg;};

        //When the user required service is not active or not exist
        exception NoSuchServiceException{string msg;};

        //When the service has already been binded
        exception ServiceAlreadyExist{string mag;};

        interface Coordinator{
                string obtainIOR(in string login, in string passwd, in string serviceName)
                raises(NotAuthorizedException, NoSuchServiceException);

                void setIOR(in string login, in string passwd, in string serviceName, in string IOR)
                raises(NotAuthorizedException, ServiceAlreadyExist);

                void clearIOR(in string login, in string passwd, in string serviceName)
                raises(NotAuthorizedException, NoSuchServiceException);
        };
};
```

Users can obtain the IOR for a service with the method obtainIOR. The user must provide the login name and password for the authentication. Privileged users, say instructors, can set the IOR for the corresponding services through the setIOR. Privileged users can also remove the service by referring the service name through the method clearIOR.

5.4 Detailed Designs and Implementations

This section will discuss details regarding designs and implementations. The overall component architecture will be given then followed by detailed discussion of each component.

5.4.1 Detailed View of Some Components of the System

Figure 5.3 shows a detailed view of some components of the system. The Event Channel module or component (interfaces as well as example of implementations) has been discussed at length in chapter 3 and this chapter. Therefore they will be touched only

briefly if they need to be included at all. In essence, the diagram illustrates components at the user side. The major components are those for chatting, drawing, image and video capture. There is a Communication Agent component or layer, which handles detailed connection to the Event Channel. The design and implementation of these components will be discussed in the following sections. Please note that some components being shown in the overall architecture are not necessarily to be illustrated in Figure 5.3.

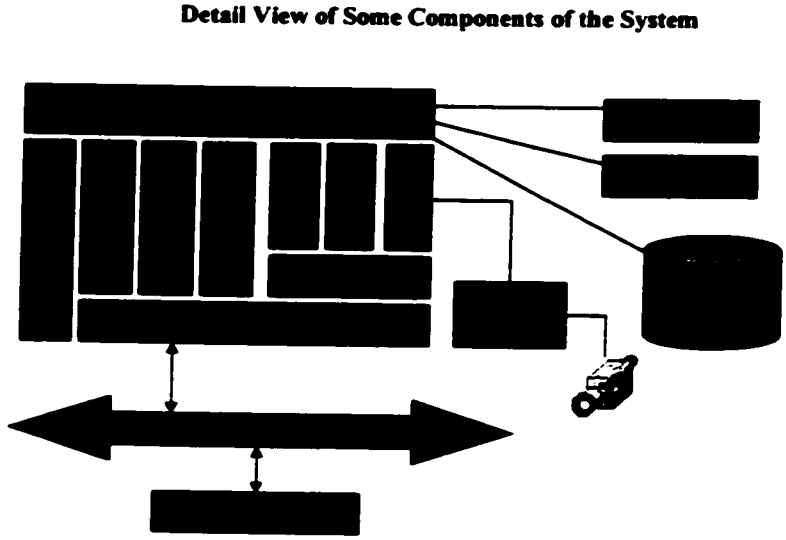

Figure 5.3 Component view for a detail design

5.4.2 Chat Component

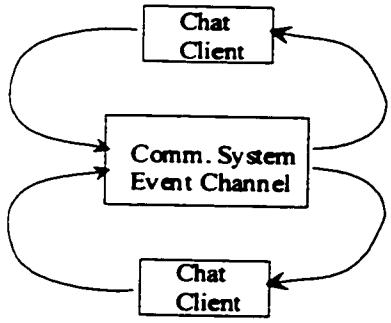

Figure 5.4 Chat component communication

With the help of Event Channel service as the central hub, the chat component could be easily constructed as shown in Figure 5.4. This over simplified architecture may be used to illustrate the basic idea and to test whether the communication is correctly functioning.

In essence, the chat client will connect to the Event Channel at both ends. As discussed previously in specification part, we should use push/push model of the Event Channel. The user can edit text in a text area. When the "Send" button is clicked, the content in the edited text area will be pushed to the Event Channel. The Event Channel will then send this message to all registered parties. The copy/cut/paste functionality is implemented so that content from other text editor can be copied to the Java text area, and vice visa. The whole chat session can be made persistent by storing to a file. In chat component, the establishment of connection to the Event Channel is simplified as a single button click—click Connect button. The detailed connecting process is handle by the communication layer, which will be discussed at section 5.4.8 Communication Component.

5.4.3 Drawing Object Component

Passing drawing objects to multi-parties is not as straightforward as that for passing text strings. The basic scenario can be described as the following. 1) A user may use a mouse to draw the drawings in the drawing area—a canvas. 2) When he finishes a drawing object, i.e. the mouse button has been released, the drawing object will be pushed to the Event Channel. 3) The Event Channel will propagate this message to all registered parties. 4) The client side will draw this object at its local canvas accordingly.
First of all we have to have a design about how the local basic drawings will be conducted. Then we will discuses the design for the functionality of this drawing component.

5.4.3.1 Classes for Basic Drawing Objects

Class java.awt.Graphics (java.awt.Graphics2D with more powerful features) provides a rich functionality of drawing methods. However, if we want to have interactive drawing process, we may as well encapsulate these drawing objects to our own defined classes. The class design diagrams are shown in Figure 5.5. These classes are in the package of classes.graphics.

The root class in this class hierarchy is the abstract class ObjectDraw, which defines a set of members and methods common to all of the drawings. The direct subclass of ObjectDraw is the class TextDraw, which encapsulates the text drawing. Another abstract class ShapeDraw being direct subclass of ObjectDraw defines some basic attributes for most of drawing shapes. The concrete subclasses of ShapeDraw are classes RectangleDraw, OvalDraw, LineDraw, and FreeHandDraw. These class diagrams emphasize the relationship among classes. Many other fields and methods have not been shown in this diagram due to the limited space in one page. The root class ObjectDraw in this hierarchy implements the Serilizable interface, implying that all drawing objects can be made persistent to a file or streams with the help of standard Java Serialization Framework. The other interface being implemented is ShapeCode interface, which is actually just a set of final static type integers corresponding to the drawing types. Under certain circumstances, we need to know the exact class type of an ObjectDraw object. We can thus use the ShapeCode information instead of using several "is instance of" queries.

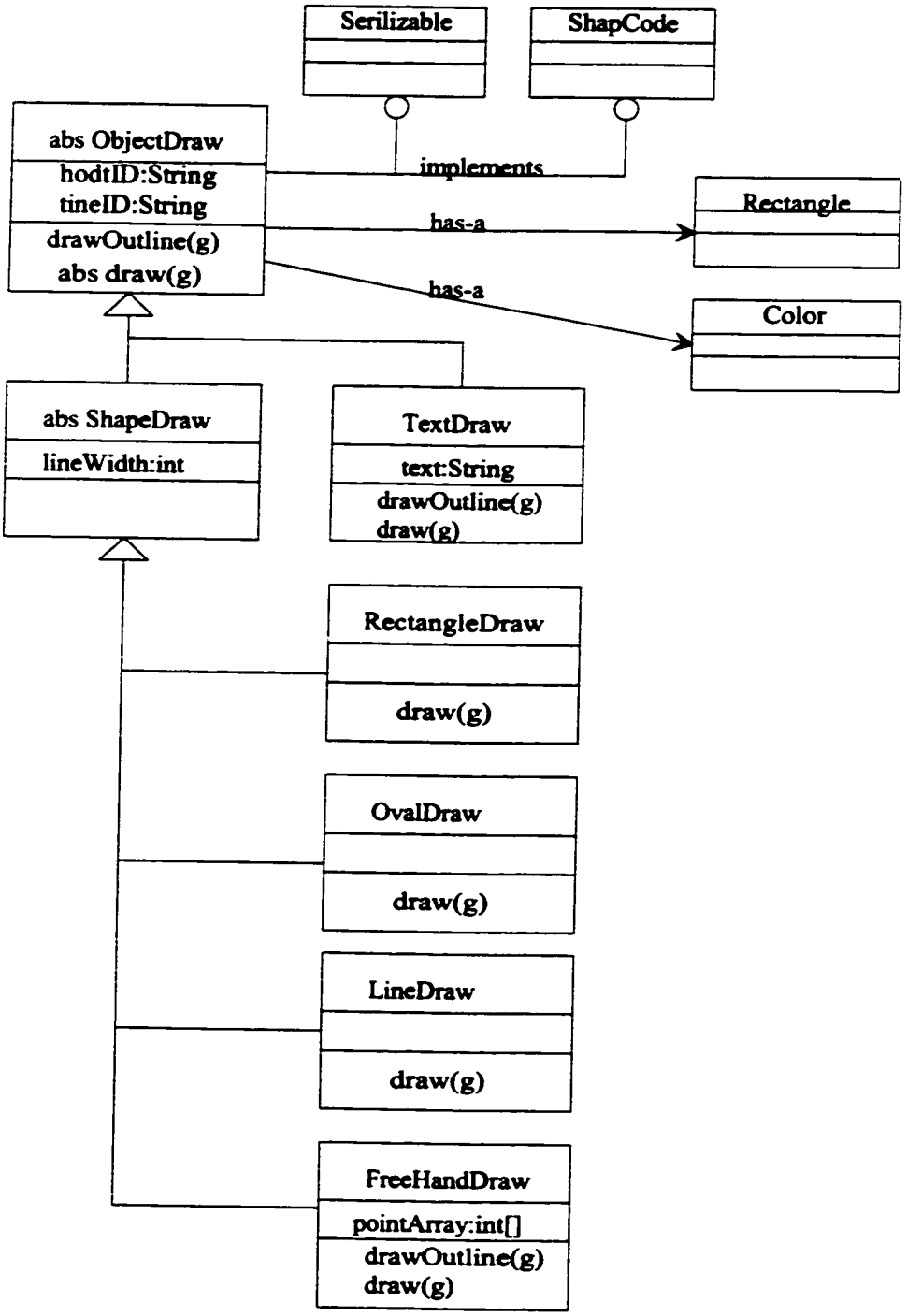

Figure 5.5 Class diagrams for drawing objects.

Global ID: You may notice two fields for IDs—hostID and timeID in the class definition of ObjectDraw. Considering the functionality for drawing update, we must have a Global ID that can be used to uniquely identify each drawing object. We devise a simple

mechanism to satisfy this requirement. The Global ID of a drawing object will be the host ID of the machine plus the time when this drawing object is created. The host ID of an Internet machine would be unique for each machine. Even in the case of the dynamic IP assignment, it should be unique in the session. The timeID is implemented as the return value of Java static method System.currentTimeMillis() call. Thus the accuracy of timeID will be to the milli-second. Since the drawing process is with a pace of the mouse moving time scale, this timeID is sufficient for our purpose.

Other basic attributes: 1) the bound rectangle—Rectangle rect, all drawings should have a boundary; 2) method drawOutline(), which draws the "rect" when being selected; 3) abstract method draw(), which should be implemented for each drawing object. We have a type ObjectDraw, which can represent all of its sub types. When its method draw() is called, the actual drawing object will be invoked. This dynamic polymorphism in Java reflects the beauty of Object Oriented Programming Language.

Based on the prototype nature of this project, the above classes have a sufficient "expressing" power in drawing. Sophisticated drawing program is a project itself. However our class design is an extensible one, which allows new drawing types to be added to the architecture by sub-typing.

5.4.3.2 Functionality for the Drawing Component

The behavior of the drawing component is much similar to that of a conventional drawing application. You can click the shape that you wish to draw. Then use the mouse to draw this shape accordingly in the canvas. You can also set the color for the drawing. The drawing object can be resized and moved around after being created. All these activities are necessarily to be reflected to drawing canvas of all parties.

Data structures for storing drawing objects: The drawing objects are stored in a Java Vector object (a kind of dynamic array). When repaint() method is called, all drawing objects stored in the Vector will be drawn one by one. Additionally, the data structure Hashtable is also used in maintaining drawing objects. The key for this Hashtable would be the Global ID→ hostID&timeID (concatenated here) and the value would be the drawing object itself. This Hashtable will provide direct access to the drawing object

based on its Global ID. This is important for the update process. If we search around the Vector for an object with a particular Global ID, it would be too inefficient. One important point here is that this updating thread corresponds to that of the remote invocation from the Event Service. An efficient algorithm in updating is thus significantly important. This justifies the use of Hashtable for this update issue. Therefore we have utilized both Vector and Hashtable data structures in storing our drawing objects. This allows possibility of both walking through and direct accessing to drawing objects.

Drawing passing process: When a drawing object is being sent away, its IDL type DrawCORBA (as defined in module jidl) will be constructed. There is a standard way to insert this DrawCORBA into an Any object then this Any object will travel through the Event Channel and arrive at the client side. Upon receiving the DrawCORBA object, the client will reconstruct the drawing object based on the parameters in DrawCORBA.

We have observed that the set of classes in the package of classes.graphics has made our local drawing process a very effective one. Unfortunately, this kind of class type can not be transferred via Event Channels that require IDL type. The IDL type, e.g. DrawCORBA, maps to a class in Java after compilation with the IDL to Java compiler. One may think that the drawing classes can subclass this IDL type, which may enable the drawing message directly pass to Event Channel. This approach poses two problems. First, if each drawing object has its own corresponding IDL type, their subclass relationships will make our local drawing a difficult task. The second problem is beyond our control. That is the Java class so generated by the JIDL compiler is a final type class, which can not be inherited. We of course could manually modify the "final" key word, but it is not advisable.

Another approach may use Java Serialization/De-Serialization process as discussed at section 5.3.3. The drawing message will be essentially the same as image messages, both being represented as byte array. Although being a cleaner approach, the serialization processes incur overhead that may takes more time than constructing IDL type. Additionally, in the implemented approach, we distinguish the original drawing objects and update ones thus reduce the unnecessary information to be sent over the Internet. Moreover pure Java solution (i.e. no CORBA component as the middleware) may have

better way to deal with this problem, because all serilizable objects can be directly transferred to Java platforms over the Internet. That is the case for both socket stream and RMI. This is the advantage of pure Java approach. But we gain many benefits for the robust server services with incorporating CORBA Event Service.

5.4.3.3 Consistency in Drawing Collaboration

Consistency issues frequently arise from the separation of processing resources and the concurrency in distributed system [Coulo96]. They include update consistency, replication consistency and cache consistency etc. When a new drawing object is being created, it does not pose a consistency problem in general. Since users can move and resize the existing drawing objects concurrently, this may potentially leave different drawings at each client canvas. Because we must ensure certain responsiveness for the local drawing, when the local action is conducted, it must be reflected immediately at local drawing canvas and then multicast this change to the other parties. Since the update messages from the Event Channels are consistent to all the clients (as discussed in chapter 3), the ultimate effect will be consistent for each client. However, the sequences of the actions in reaching the final effect are not necessarily the same to the view of each client.

5.4.4 Images Handling Component

This section is dealing with images handling component. We will discuss the possible means in effectively transfer the image messages over the Internet, including the image compression. Then we will discuss about the implemented functionality of this component.

5.4.4.1 How to Pass Image Messages

To devise an effective means to transfer image messages over network is not a so trivial task. In Java environment, there is a powerful writeObject() for I/O, which can write all serializable object into a file or a byte stream. For example, we can write String or Vector objects by this way. The Java Serialization Framework will take care all of the details. However the Image type in Java can not be handled in this way. It is also not feasible to have an IDL type defined to represent the Image type.

In fact, the image in the memory is simply a two-dimensional array or a one-dimension array with pixelsOfWidth*pixelsOfHeight elements. The value of elements in this array is the RGB (24 bits color representation of Red, Green and Blue in RGB color model) value corresponding to each pixel. For a very simple approach, we can just retrieve this two-dimensional array from the memory and send it over as IDL octet sequence. This IDL type has been defined previously. The receiver will reconstruct this image with available Java graphics facility. We actually tested this approach and it worked smoothly. However we encountered performance problems for network traffic. Considering a simple 320x240 image with 24bit color for each pixel, the size for this uncompressed image would be 230Kb. To send this amount of data to multiple parties over the Internet would consume significant amount of network bandwidth. Therefore images must be compressed before they are being sent over the network.

Compressing the image:

The most popular compressed formats for image are .gif (or GIF) and .jpg (or JPG, JPEG) in the Internet world. While GIF may be good for cartoon type image compression, JPG format is very effective for compression of pictures. It is very easy to achieve a factor of 10 of compression ratio for JPG format.

Therefore we will choose the .jpg format for our image message passing. The components to compress image into the compressed format and to construct an image from the compressed format are termed as coder and decoder respectively. They are together referred as the term of codec.

Codec (or Coder and Decoder) could be either a hardware or software component. Obviously we are talking about software at this time. Sun has a package called "com.sun.image.codec.jpeg", which provides functionality we need. But it is not part of Standard Java Platform. This is a kind of framework for image codec purpose. Therefore it is a bit too heavy to bundle this package with our distribution. Additionally the package itself is a bit too complicated to walk through. As a matter of fact, AWT package has already a standard way to read a .jpg (as well as .gif) file into the memory. Therefore we just need to provide a means to compress image in the memory to a .jpg format.

A Java JpegEncoder component is available, which resides in the package of classes.jpeg. This is largely a rewriting in Java from its C code counterpart. The use of JpegEncoder class is pretty straightforward. We just need to provide an OutputStream and a handle for an Image. There is an additional parameter called Quality, which is ranged from 0 to 100 (from poor image quality, high compression to good image quality, low compression). To achieve higher compression may take much longer time (kind of exponential growth). We set the default Quality=50, which has been tested to have a good balance between the compression ratio and the time to conduct this compression. The compression process will be irrelevant if the time spent on this task is more than that to send its original message over the network (we are concerned with the dynamic compression on the fly).

5.4.4.2 Functionality of Image Component

We assume that users will view images first then will decide whether to send them over. The images with either .jpg or .gif format can be loaded into Java environment by standard Toolkit in Java application or getImage() in Java Applet. Such image can be drawn on a Canvas. When the user click the button "Send Image" (presume that the user has got an handle to the ProxyPushConsumer at the Event Channel), the image will be compressed and sent to the Event Channel as the way discussed at last section.
Additionally, multiple images can be loaded into the memory and sent as a group. More accurately, a group of images can be sent one by one at a certain rate. In fact a whole directory of images files can be loaded into the RAM. Users can set the pushing rate interactively. The feature of this multiple pushing can be used as a way of simulating the animation or the slide show. It is a useful feature for the prepared presenters. This functionality can also be used in testing the performance of network traffic for continues pushing of captured images without employing an online camera.

5.4.5 Real-Time Video Capture

The interface to the Communication System (Event Channels) does not pose any differences among different types of media, being either text, drawing objectS or images. We will focus on the design for the video capture facility at this part. Once the image is

captured, it will be pushed to the communication system just as the same way with normal images.

Some considerations: 1) This capture facility may be provided only to the privilege users e.g. instructorS. For each collaboration group, there are one captured image push Producer and many Consumers—classroom-like style. Therefore client side can have one window or one canvas to be associated with this media. 2) Of course, the number of the producer could be more than one—communication system does not care. This is similar to videoconference. But in the client side it is necessary to have multi-windows, where each window corresponds to one producer. Although it is possible to have such functionality, we will limit one image capture pushing for one Event Channel in this prototype. 3) This capture functionality provides one more dimension to the means of input for instructors.

5.4.5.1 Non-Integrated/Loosely Integrated Approach

At the start of this work, Java Environment has Java media framework JMF1.0, which provides functionality of displaying video or audio. However until recently, the full release of JMF2.0 provides video capture facility. Before this time, there is no easy way for Java directly access camera. Even with this capability in JMF2.0, the pure Java solution is still lack of certain performance. This is because the video capture is a very resource intensive activity. For performance enhancement, Sun provides as well platform specific reference implementation, which is a 5Mb package for Win32.

On the other hand, C++ application has libraries readily available for the access to video camera. For windows, the standard API for accessing video-camera devices is VFW Video for Windows. A straightforward solution to incorporate the video capture facility is schematically shown in the figure.

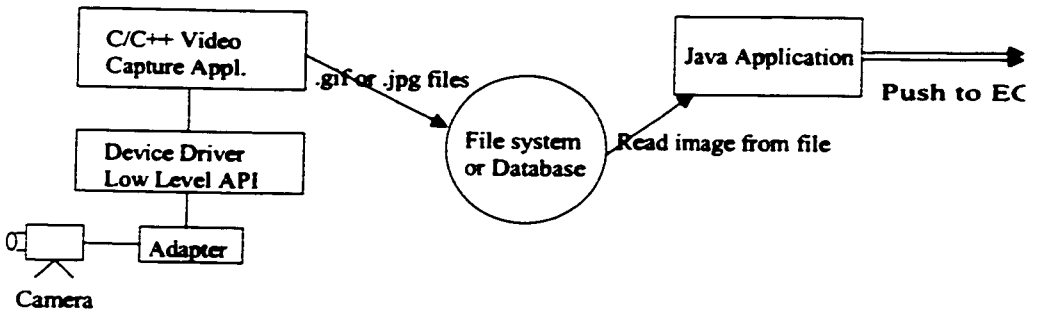

Figure 5.6 Non-integrated approach for video capture and Java Application

This is the so-called non-integrated or loosely integrated approach. This is that the Java application and video capture application are virtually separated entities. Figure 5.6 illustrates a possible architecture of this approach. Although being simple, the immediate problems could be 1) it may need two UI interfaces and two isolated applications, which is awkward to work with; 2) it may be enough for occasional off-line capture, but is not suitable for the online real-time capture.

We have actually tested this approach. The video capture application utilized is simply the one provided with the Camera. We set a certain capture rate, say one image per second, and save the capture the image to a designated place i.e. the same file location. A Java application will read this file with certain rate and push this content for the communication. There are two major problems. First of all, to save and read the content in the second storage is extremely expensive in compare to the memory operation. Perhaps it is even slower than that of network traffic. Secondly, because there is no easy synchronization mechanism between these two applications, they can be in conflict in accessing the file. It happens occasionally that either the capture can not proceed or the Java application read a corrupted file. In either case, the whole process will be terminated unexpectedly. Therefore other approaches should be pursued in order to solve these problems.

5.4.5.2 Fully Integrated Approach

The fully integrated approach is to use Java Native Interface, JNI to connect the Java application part with the C/C++ video capture part [Marti99]. The architecture of this approach is shown in Figure 5.7. One of the important interface in JNI is to have pixels

copied from C++ memory space to Java VM's memory space. These operations are conducted in the main memory, thus are much more efficient than those with I/O operation involved. In addition, the threads in both Java and C++ could now be coordinated.

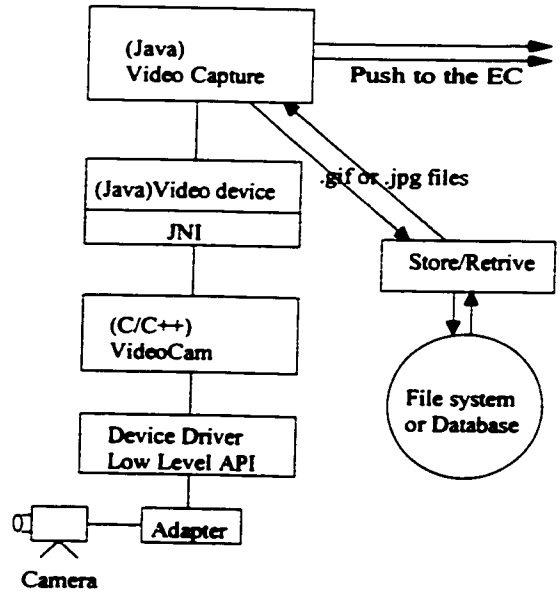

Figure 5.7 Fully integrated approach for video capture and Java Application

Copying C Array to Java VM and Image Format: The C++ part in this component is using VFW, standard video capture libraries. To write an application against the VFW interface will not only save the development time but also portable to many device drivers. This has been Microsoft standard for Windows. It is worth noting that there is a mismatch between Windows 24-bit image format (BGR upside-down) and Java byte order (RGB right-side up). Therefore the conversion must be made when copy the C array to the Java array of pixels. If it has to be a frame-by-frame transfer, the transfer speed has to be optimized. Therefore the conversion between the image formats would be conducted in the native implementation to have better efficiency.

Thread synchronization: Many low-level APIs in image capture employ Callback functions, which requires synchronizing multi threads in both Java and C++. In Java, we have synchronization mechanism to coordinate multi-thread. In VC++, we can use

semaphore to coordinate multi-thread through the library call of WaitForSemaphore() and ReleaseSemaphore(), etc. For example Java application initializes a method to scan a frame, it will invoke the native scan method through JNI interface. In native C++ part, it will issue capGrabFram() method to instruct the low level APIs to grab the frame. Windows environment will use Callback to return the grabbed result, which is an asynchronous process. These threads would be synchronized with semaphore. This is that the action to copy C array to Java VM space must wait the corresponding Callback to complete.

Java images presentation: The design of this part is reflecting the principal of decoupled and asynchronous approach. The image will be presented at a canvas that implements the interface Observer. The image grabbing class VideoGrabber extends the class of Observable. Observer and Observable are standard Java interface and class respectively. The Observable object can notify all registered Observers that certain changes have occurred. The Observers can do the update() accordingly. Therefore the image presentation part does not need to wait synchronously for new event or actively poll the results. It will be notified when the new array of pixels is arriving.

Captured images pushed away: Once an image has been grabbed into JavaVM space, it is straightforward to push it into the Event Channel (presume that the user has got a handle to the ProxyPushConsumer at the Event Channel). The class VideoMediator residing at the package of classes.camera will encapsulate the functionality of conducting image capture and sending the captured image simultaneously. The capture/pushing rate and size of the frame can be set interactively by the user.

5.4.6 Media (Video/Audio) Playing Component

With the help of JMF, both video and audio clips can be played. The classes MediaPanel and PlayerFrame residing at the package of classes.swing are primarily responsible for invoking the playing functionality. Users can choose either play the local media file, or download from the Internet, or even the lively captured one. (The live captured media playing with JMF is not implemented in the current work.)

The MediaPanel uses PlayerFrame to interface a Java Bean MediaPlayer, which is distributed with JMF2.0. MediaPlayer enables a broad rang of media formats to be played. For the details regarding the compatible formats please refer to the JMF2.0 specification document. PlayerFrame implementing the interface ControllerListener is listening to the Media Realized Event. At the Realized state, the visual display component and control panel component can then be retrieved form MediaPlayer by calling getVisualComponent() and getControlPanelComponent() methods. The control is also passed to the MediaPanel.

The MediaPanel incorporating MediaPlayer Java Bean requires JMF2.0 support. The library size for JMF2.0 cross platform pure Java implementation is about 2.5Mb. If this distribution is a bit too much, we can replace MediaPanel with AudioPanel. These two panels have exactly the same GUI outlook, thus ensuring a consistent look. But AudioPanel does not need JMF2.0 support. AudioPanel just use the basic audio clip playing facilities in Java standard distribution with rather limited functionality.

5.4.7 Utilities Package

The utility classes are resides at the package of classes.util. There are some handy pre-build dialogs ready to be used. One important component in the utility package is the MailAgent. This is a handy SMTP (Simple Mail Transport Protocol) Java Application for simple email. This email client utility provides users an additional dimension for interaction besides the already rich ways in real-time communications. A user can create an instance of this email client at any point along with his session. The default port for SMTP is 25. You can specify any server machine that supports SMTP. Sometimes it involves security problem with firewall. For example, if you specify the server machine to be "server.uwidnsor.ca", you have to be within the Campus. You are limited with text content for SMTP though. If an email has been sent successfully, its content will be saved to the file you specified. It also supports nickname email address matching.

Another useful class in this package is IORFile. It provides two static methods for writing IOR to a named file or retrieving the IOR from the file. A user can also invoke a web browser at any point of his session by click a menu item.

5.4.8 Communication Component

The communication component resides at the package of classes.comm. The classes of PushConsumer_impl, PushSupplier_impl, PullConsumer_impl and PullSupplier_impl are the implementations of IDL interfaces PushConsumer, PushSupplier, PullConsumer, PullSupplier at module COSEvent, respectively.

The module CosEventChannelAdmin implementation is provided at the package com.ooc.CosEventChannelAdmin including the implementation for the interfaces ProxyPushConsumer, ProxyPullSupplier, ProxyPullConsumer, ProxyPushSupplier, ConsumerAdmin, SupplierAdmin, and EventChannel. They are bundled with ORBacus3.2 distribution.

Although the CORBA Event Service IDLs are well written, it takes some time to get familiar with the effective use of it. People complaints that it takes three steps in connecting to a Event Channel [Schmi97b], i.e. 1) Using EventChannel to obtain SupplierAdmin; 2) Using supplierAdmin to obtain ProxyPushConsumer; 3) Constructing PushSupplier with this consumer and connecting the consumer and supplier as necessary.

To make a friendly use of these interfaces, CommAgent class is implemented to abstract the use of the IDL interfaces. Two important methods in CommAgent are getSupplier():PushSupplier_impl and setConsumer(transObj:TransObject). The method getSupplier() will do all the connection work and return a handle of PushSupplier_impl to the caller. The caller can use this handle to push messages to the Event Channel. The method setConsumer() will do all the necessary connection work and to activate the consumer object in a separate thread. It is important to have a separate thread when we have both supplier and consumer in the same Java VM.

When we use CommAgent in our system, we assume users will primarily connect to the Event Channel at both ends. Therefore when a user clicks the button "Connect", both getSupplier() and setConsumer() will be invoked thus completing the connection at both ways.

There is an interface called TransObject which contains a single method update(Any any). This "any" is the object received by the consumer from the Event Channel. The

component wishing to interpret the received "any" should implement the TransObject interface. In our system, the class CyberClassRoom implements the TransObject interface. Upon the update, the CyberClassRoom will update its content in its chatting, drawing or imaging sub components accordingly.

The diconnect() method inside class CommAgent will effectively disconnect both supplier and consumer ends. At the client GUI, a toggle-like button will connect and disconnect Event Channel by consecutively clicking. The BOA instance will not be destroyed for each of disconnecting invocation, where only corresponding CORBA objects are deactivated.

Another class MyORBInit also in this package perhaps is worth noting, which consists of two static method getORB() and getBOA() to return orb and boa instances respectively. This class maintains a singleton ORB initialization within a single Java Virtual Machine. We have experienced problems, for example when more than one of ORB init() at different point with one virtual machine, BOA initialization exception will be thrown at the run time. That is why MyORBInit is at its place.

5.4.9 Coordinator Component

Coordinator component is at the package of coordinator, which is essentially comprised of implementations of IDL interfaces for coordinator module. The root of this package is not started with "classes", our root for many previous discussed packages. This is because Coordinator is a logically and physically separated entity from those classes. There are 17 Java source files inside the coordinator directory, in which majorities are for the stubs, skeletons and helper files. The class Coordinator_impl implements the Coordinator interface as prescribed at section 5.3.3. As we discussed previously, this component will incorporate Oracle8i DBMS through JDBC. We thus discuss each part in the following.

5.4.9.1 JDBC Setup for Oracle8i

JDBC requires appropriate database driver to access a database. After this driver is setup properly, the JDBC call is just straightforward. We can normally employs JDBC/ODBC driver provided with JDBC package. This JDBC/ODBC driver provides solutions to a

class of database connection when ODBC is supported by the database. The trend is to use the direct JDBC driver provided by the Vendor, which shall remove the overhead for JDBC/ODBC layer. Oracle8i has a JDBC driver and the commander to set it up is as the following segment:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
```

Additionally, the connect string to the database instance for this system is as the following:

```
url="jdbc:oracle:thin:@3139erie-13.lams.uwindsor.ca:1521:arjDB"
```

You may notice that this is a fully Internet accessible path. Therefore the middleware Coordinator can reside at a different machine other than the one Oracle DBMS is running despite the fact we run them at the same machine.

When the middleware Coordinator is running, the database connection is maintained. This can increase the performance because establishing the connection to Oracle Server incurs expensive overhead. But it may not be appropriate for resource and security reasons to keep the connection alive extensively. In the actual use, the policy can be decided to make a balance between the performance and resource utilization.

We can connect, for instance, to the University database to obtain the information for authentication process. To facilitate the prototype use, we have designed some simple tables for the demonstration purpose only. These tables include NormalUser, PriviligeUser, UserInfo, CourseComm, Enrollment, etc. NormalUser and PrivilegeUser tables are storing login ID and password for average and privilege users respectively. CourseComm table will store the service name (e.g. Event Channel for the course), its corresponding IOR and other fields such as the session start time and status. Furthermore, Enrollment table will store students' registered courses.

5.4.9.2 Coordinator Implementation Consideration

Since the Coordinator component is essential for the authentication, in order to obtain the IOR for certain services, we need to have the IOR of the Coordinator server to be distributed to the clients. Therefore this IOR must be made persistent. This is that each time the restart of the Coordinator will have the same object reference. To make this happening in ORBacus, the instance of Coordinator must connect with the same name for example "Coordinator". The commander is as the following,
((com.ooc.CORBA.ORB)orb).connect(coordinator, "Coordinator"). Additionally you must start the Coordinator server for a specified port.
There are two files called Coordinator_Client and Coordinator_Server also associated with this package. Coordiantor_Server is simply a server process to make Coordinater available to be remotely accessed. Coordinator_Client will be distributed to the client side, which makes the access to the Coordinator easily.

5.4.10 Dynamic Event Channels Creation and Registration

In our system architecture, users can create Event Channels dynamically and interactively. Users can choose to register the newly generated channels either to Naming Service or to the Coordinator. To facilitate dynamic channel generation, there must be some EventChannelFactory instances running in the distributed system. A user can definitely start his own. We will use human readable "iioploc" approach [Vinos98] to resolve these EventChannelFactory's object references instead of using IOR. In the system all EventChannelFactory will be running at the port of 8000 with an object key of DefaultEventFactory. For example, EventChannelFactory in our NT workstation will have IOR equivalent—iioploc://137.207.16.20:8000/DefaultEventFactory.
The caller to create a Event Channel with EventChannelFactory is through class EventFactoryDialog residing at classes.swing package. There are dialogs—RegisterNamingDialog, EventFromNamingDialog, and LogDialog, encapsulating functionality to register (bind/unbind) to Naming service, resolve Event Channel from Naming, and register (set/obtain IOR) to Coordinator respectively. The naming service is with a IOR associated with iioploc://137.207.16.20:5000/NameService.

5.4.11 Incorporating Web Server and Servlet

It is desirable that our system framework will interact with the Web Server since web is a perfect vehicle for delivering information. While the Web Server Apache1.3.6 installed in this machine provides general information to the public, a small Java Server provided by JSDK2.0 enables Java Servlet, listening at the port number of 8080. This Java Servlet can communicate with Oracle DBMS through JDBC. At the minimum use, users can inquiry about the available event services through the web. Then they can use this information e.g. the service name to get the IOR from the Coordinator middleware. For the future enhancement, administrator shall be able to do the administrative work through this Servlet/JDBC/Oracle architecture over the Web.

5.5 System Features and System Requirements

In this section, system features will be described. Although various aspects of the system have been discussed here and there in this thesis so far, this section provides a summary of features including those not having been explicitly addressed before. Furthermore the system requirements for both server and client side will be presented in order to assist the use of it when it is deployed.

5.5.1 System Features

In essence, the system will enable users to communicate and collaborate online in real-time with functionality of chatting, white boarding, image pushing, video capturing etc. The detail features are listed as the following.

- **Chatting:** Users can have group chat within a certain channel. Users can edit chat input or use cut-paste functionality to transfer the text from other text editors. Upon connecting to certain Event Channel, a user can send messages to and receive messages from this channel. The whole session can be saved and retrieved at the local storage.

- **Drawing:** Various shapes such as rectangle, oval, line, text and freehand drawings can be drawn on the canvas associated with the draw panel. The functionality is similar to a mini-paint program. When a user draws a drawing-object, the same drawing will appear at the screen of all users. When a drawing object is being resized or moved, the same effect will be reflected to all of the clients. Drawing panel can be set as being either inserted within the whole GUI or floated. The whole drawing session can be saved and retrieved.
- **Image:** Images can be loaded, saved or sent to all partners. It supports both GIF and JPEG formats for loading images. When images are saved or sent, the format supported is JPEG only. The compression ratio can be set in order to accommodate the different requirements for the tradeoff between the image quality and the file size (or more importantly, the network traffic). There is a multi-load mode, which can load image files of a whole directory to the buffer. The Send-All can send all images in the buffer to all the clients with user settable rate (5fps, 2fps, 1fps etc.). Images in the buffer can be browsed so a user can select the desirable images to be saved to the persistent storage.
- **Real-time video capture:** Video images can be captured and be sent to all of the parties at the real-time. The pushing rate and resolution of images can be set at the run time. This feature may be reserved to the privileged users, say instructors. If all parties are able to send real-time captured images, it will be too traffic consuming in the Internet environment.
- **Video and audio playing:** Video and audio clips can be played. The clips can reside either in the local file or downloadable over the Internet (http web server and other media servers).
- **Authentication:** All users including students and instructors are required to go thorough the login process before they are able to use the facility for communication.
- **Persistent storage:** All chatting, drawing sessions and images can be stored and retrieved at local storage. There could be an option to store and retrieve sessions from the center database. This option is desirable if local storage is not accessible (for example when Applet is treated as un-trusted).

- **Dynamic Event Channels creation:** A user can start a new Event Channel and register its name and IOR to a middleware, Coordinator. That means the system can be dynamically reconfigured. Alternatively, dynamically created channels can be registered to the Naming Service, which does not need going through the authentication process.
- **Utilities:** A Light weight Java E-mail component developed for this project can be triggered when a user wants to send an E-mail as well. The web browser can also be started from triggering browser within the application. This is kind of integration with the web browser (loosely coupled though).

5.5.2 System Requirements

We will list the requirements for both Client side and Server side. Due to the distributed object nature of the system, there is no distinct line in specifying the Client and Server sides. In particular, Event Channels can be started at any client machine. When we talk about the server side, we mean that the machine running Oracle Database, Web servers as well as CORBA middleware. The client side will be all the rest including both student-like users and instructor-like users.

- **Client Side:**

  Hardware:

  - A PC with Pentium 90Mhz CPU or equivalent, 32 megabytes RAM minimum, 48 megabytes RAM recommended that is the requirement for Java 2 platform runtime.
  - If video capture is required, a PC with 166MHz Pentium CPU and 64 MB RAM are required. A video camera or capture card (e.g. QuickCam VC or other Window compatible capture facilities.) is required.
  - A 28.8kbps modem or LAN network card connected to Internet.

  Software:

  - Java 2 platform runtime.
  - ORBacus3.2 for Java
  - Our distributed communication package

- JMF2.0, if video/audio playing required
- Web browser (Netscape Navigator4.x or Microsoft Internet Explorer4.x)

- **Server Side:**

  Hardware:

  - A PC with Pentium 233Mhz CPU or equivalent, 96 megabytes RAM minimum, running on NT4.0 operating system
  - LAN network card connected to Internet.

  Software:

  - Java 2 platform runtime.
  - ORBacus3.2 for Java
  - Our distributed communication package
  - Oracle8i, Database Manage System
  - Web Server (e.g. Apache 1.3.6) and JSDK2.0 for Servlet

5.6 The Operation of the System

Server Start:

We will list services that must be presented in the distributed system and describe the sequence of starting these services.

Oracle database instance arjDB and Web Server are running as NT Services (daemon processes).

To start Coordinator:

```
java coordinator.Coordinator_Server -OAport 20001
```

To start Naming Service:

```
java com.ooc.CosNaming.Server -OAport 5000
```

To start Event Channel Factory Server:

```
java classes.server.Server -OAport 8000
```

This Event Channel Factory Server will enable users to dynamically generate event channels, which can be registered to the Naming Service or to the Coordinator according to the users' choice. However if you want to start a default event channel whose reference has been distributed with the package, you can type

java classes.server.Server -ORBservice EventService iiop://137.207.16.20:5001/Event --ior

If you need the IOR to be printed to a file Event.ref, you may type the following:

java classes.server.Server -ORBservice EventService iiop://137.207.16.20:5001/Event -f Event.ref

Client Side:

At each client console, type the following

java classes.swing.CyberClassRoom

The client is now ready for the interactive communication.

The system has been firstly tested over the university Intranet. Server services and Event Channel are running at our NT4.0 workstation. Two client instances are running at this machine as well, one of which is associated with image capture and pushing actions with a setting rate 1fps. Up to four more clients are running at UNIX machines X-terminal clients, i.e. Marigold and Montague, two UNIX machines in the School of Computer Science. The reason to use the UNIX clients is because we have many X-terminals available around in the Computer lab. It is easy to have a few terminals than to have a few PCs for dispose. On the other hand, it also demonstrates that the interoperability across different operating system. The overall performance is very satisfactory. The captured images can be effectively propagated to all clients. At the mean time chatting and drawing can be conducted. That is the case that all three different media types—image, drawing object, and text can be effectively communicated concurrently over our communication system.

The system has also been tested with clients running Win95 over the Internet with 28.8kbps-modem connection. The chatting and drawing communication are light communication in nature, thus pose no any significant network delay. The image communication can also be effectively conducted, but the push rate should be kept low in compare to the Intranet circumstance.

It is worth noting that the university firewall does not block our CORBA communication at all. However if the firewall ever impose the restriction on the traffic other than E-mail or Web-related ones, the HTTP tunneling approach could be placed [Vogel97]. But this

HTTP tunneling will incur so much overheads for the communication, which may make the real-time communication impossible.

5.7 Rooms for Improvement

There are sufficient functionality and capacity for this prototyping of the overall system. However there are many places that could be improved regarding issues of either the performance or design alternatives.

1) Applet version development

We have implemented our prototype using Java JDK1.2 and there are currently no popular web browsers capable of running java1.2 Applet. It is possible to modify our Java application into Java Applet. A trusted Applet will be employed if disk operation is required. The advantage of using Java Applet is that the Applet can be downloaded dynamically by the Web browser. Therefore the client can use our system at any point in the Internet without pre-downloading our package. The cost of distribution and maintenance will be reduced significantly. The disadvantage is that the Applet download and start take very long time especially if we have to bundle the support software (ORBs and Services etc.). Please note that this Applet version will not consist of the video capture functionality as it has been implemented as native method. Additionally, Applet has an inhabitant limitation regarding the socket connection, which is restricted to the server machine where the Applet is downloaded.

2) Allocate a dedicated channel for each media

We could for example allocate three channels for one group communication with each channel for one media. This allocation should be transparent to the users. For a more rational arrangement, we can allocate one channel for image and one channel for text and drawing. The continuously captured images being sent over the Internet will consume much network bandwidth. On the other hand the missing of the images has higher tolerability than that of drawing or chatting. We can simply make a smaller length of queue for the image channel.

3) Balancing the channel load for better salability

Additionally, new event channels could be automatically activated on demand in different machines. At the certain point (when reaching certain numbers of the client), the output of the event channel could redirect to the newly created event channel and this channel is further connected to the broadcasting end of adding clients. Ideally the event service will be located at different network. Therefore both the machine load and network load will be redistributed.

4) Using C++ CORBA object for the Event Channel.

If the event channel becomes a bottleneck for the system, C++ event channel may replace the Java event channel to enhance the performance. It is known that C++ native code can outperform the Java counterpart, even with JIT compiler being placed. The CORBA IIOP [OMG98a] will guarantee their interoperability. This underscores the beauty of CBD.

5) Using JMF to capture video/audio

JMF 2.0 enables sound/video capture, which is the standard Java solution. We currently just use its playing facilities. Although capturing with JMF2.0 poses certain performance concern, the eventually cheaper computer hardware will make this graceful Java solution even more attractive. This is that we would have a complete Java solution.

6. Conclusions

This thesis is concerned with the online real-time multi-party communication and collaboration. We employed distributed object approach, which has higher abstraction than that of the TCP/IP socket. We follow the component-bases development principles, which emphasize utilizing existing software components and standard services instead of building systems from scratch. A novel approach in utilizing CORBA Basic Object Service, Event Channel as our message exchanging central hub in real-time communication was also devised. In addition to the Object Naming Service in helping to locate the distributed objects, an approach to authentication process, incorporating database manage system has been an indispensable part of our distributed system.

The overall distributed system has been formally specified with the rigorous mathematical modeling. This modeling describes the essential components and their behaviors that must be exhibited in such distributed system. This modeling shall enable software engineers to observe and examine the behavior of the distributed system under development.

Based on this specification, a feature-rich prototype for such distributed systems has been developed to validate our approach. This distributed system supports a range of media types (e.g. strings, drawings and captured or pre-stored images) real-time online communication and collaboration in a uniformed manner. Although being developed primarily for the online education system, the system can be used largely in any real-time online communication and collaboration systems.

Running tests of such distributed system in an Intranet and Internet environment have demonstrated its capability and functionality in an effective online real-time communication and collaboration for a class of media types.

- **Summary of Contributions:**

- Development of an effective approach for online real-time communication. This approach is based on the principles of component-based development and distributed system design.
- Rigorous mathematical modeling with formal notation of CCS applied to the formal specification of the system. This model will enable software engineers to reason about and examine the behavior of the system to be built. This is the first time that the formal method is applied to the distance education system.
- Based on the approach proposed in this thesis, a prototype of Object-Based Online Real-time Communication System for Distance Education has been developed, which validates the effectiveness of our approach. This system is useful for a range of media communications in a real-time.

- **Future Works:**

As far as future works are concerned, there are a number of issues that could be pursued.

- From the horizontal perspective, our approach could be extended for a wide range of common real-time communication system. From the vertical perspective, our developed system is just one subsystem of overall online distance education system. The open architecture of our component-based approach will allow the system extended in various ways. The underline component infrastructure (CORBA and CORBA Services) will make such extension a reality. Therefore we can apply the approach and models developed in this thesis to an overall distance education system.
- A translation tool could be developed, which can translate the specification to CWB format. So that software engineer can conduct model checking and system behavior examining with CWB.
- As discussed in the last chapter, there are immediately improvement works regarding the implemented system itself. Such improvements will yield a system with better availability, scalability and performance.

Appendix A: List of Packages and Files in the system

We list herewith the organization of class and source files for this system. We have the top directory $TOP, which is the root for this file hierarchy ($TOP=d:\ylshaoT at the current configuration). The files are organized with directories, while they are represented in the following with appropriate indention. Normally each directory of the source files corresponds to a Java Package. Please note that the source files are not necessarily stored in the same directory as Java classes package. To be clarified, we write down the corresponding Java class package for each source file directory. For example, the directory $TOP/source/graphics corresponds to the package of classes.graphics and the source file, source/graphics/FreeHandDraw.java will be compiled into the class file, classes.graphics.FreeHandDraw.class. Each Java source file may correspond to several classes, including inner classes setting. Some directory names, e.g. graphics and swing, have the identical ones with JDK libraries. They are packages belonging to our system, which should not be confused with Java built-in library.

Java Files:

Directory source:

Directory Graphics:--package classes.graphics

File: FreeHandDraw.java, LineDraw.java, ObjectDraw.java, OvalDraw.java, RectangleDraw.java, TextDraw.java

Dirctory Jpeg:-- package classes.jpeg

File: JpegEcoder.java

Directory Server:--package classes.server

File: EventFactoryServer.java, Server.java

Directory Swing:--package classes.swing

File: AudioPanel.java, CyberClassRoom.java, DrawCanvas.java, DrawPanel.java, EventFactoyDialog.java, EventFromNamingDialog.java, ImagCanvas.java, LogDialog.java, MediaPanelDialog.java, PassDIValue.java, PassValue.java, PlayerApp.java, PalyerFrame.java, RegisterNameDialog.java,

ReValueDialog.java, ShapeState.java, StartEventDailog.java,
SwingImageCanvas.java, VideoCanvas.java, VideoPanel.java

Director Util:--package classes.util

File: AboutDI.java, ImageCanvas.java, IORFile.java, MailAgent.java

Directory Comm:--package classes.comm

File: CommAgent.java, CurrentIOR.java, ImagePushAgent.java,
MyORBInit.java, PullConsumer_impl.java, PullSupplier_impl.java
PushConsumer_impl.java, PushSupplier.java, TransObject.java

Directory Camera:--package classes.camera

File: VideoGrabber.java, VideoGrabberCanvas.java, VideoMediator.java,
Win32VideoDevice.java, BasicSettingPanel.java,
ControlPanelFrame.java, ConnectFailedException.java,
DisconnectFailedException.java

Directory Coordinator:--package coordinator

File: Coordinator.java, Coordinator_Client.java, Coordinator_impl.java,
Coordinator_Server.java, NoSuchServiceException.java,
NotAuthorizedException.java, ServiceAlreadyExist.java,
StubForCoordinator.java, and xxxHelper.java, xxxHolder.java etc.

Directory jidl:--package jidl

File: DrawCORBA.java, DrawCORBAHelper.java, DrawCORBAHolder.java
UpdateCORBA.java, UpdateCORBAHelper.java,
UpdateCORBA.Holder.java,
PointCORBA.java, PointCORBAHelper.java, PointCORBAHolder.java,
PointSequenceHelper.java, PointSequenceHolder.java,
ByteArrayHelper.java

Compile Java files:

At the $TOP directory, for example if we want to compile DrawPanel.java,
we have to type the following:

```
javac –d . source/swing/DrawPanel.java
```

IDL Files:

File: ByteArray.idl:--module jidl, Coordinator.idl:--module coordianator,

Draw.idl:--modual jidl,

Compile IDL file to Java Source File:

At the $TOP directory, for example if we want to compile ByteArray.idl, we have to type the following:

```
jidl ByteArray.idl
```

C++ and header Files:

Directory source/swing/camera:

File: classes_camera_VideoDevice.h, VideoCam.h, VideoCam.cpp, VideoDevice.cpp

Compile JNI to C header file and C++ file to DLL:

For JNI automatically generated header file (classes_camera_VideoDevice.h) At the directory where VideoDevice.class is located, type the following:

```
javah -jni VideoDevice
```

To create DLL library (grabber32.dll), we have to type the following:

```
cl -LD VideoDevice.cpp VideoCam.cpp kernel32.lib winmm.lib wsock32.lib vfw32.lib User32.lib jvm.lib -Fegrabber32.dll
```

Appendix B: System Environmental Variables Settings

The following is an instance of environment settings at our NT Workstation. These settings are working environment when implementation is under development.

PATH=C:\jdk1.2.1\bin\; C:\orb3.2\util; C:\javaext\lib; C:\WINNT\system32; c:\WINNT;
E:\VS\MSDev98\Bin; E:\VS\Tools; E:\Microsoft Visual Studio\VC98\bin; E:\VS\MSDev98\Bin;

SET CLASSPATH=D:\oracle\ora81\jdbc\lib\816classes12.zip; C:\jmf2.0\lib\jmf.jar;
C:\jmf2.0\lib\sound.jar;
C:\jmf2.0\lib\mediaplayer.jar; C:\jmf2.0\lib\multiplayer.jar; C:\ORB3.2\OB.jar; C:\ORB3.2\OBNaming.jar;
C:\ORB3.2\OBEvent.jar; C:\ORB3.2\OBProperty.jar; C:\ORB3.2\OBTest.jar;

SET LIB=D:\oracle\ora81\lib; E:\Microsoft Visual Studio\VC98\lib\; E:\Microsoft Visual
Studio\VC98\lib\vfw32.lib; C:\jdk1.2.1\lib
SET INCLUDE=C:\jdk1.2.1\include; C:\jdk1.2.1\include\win32; E:\Microsoft Visual Studio\VC98\include

These settings will ensure all of our commands presented at Appendix A as well as those in the text body executed adequately. However these settings depend on where the corresponding software located. Apart from variables for PATH and CLASSPATH, there are variables for LIB and INCLUDE, which are required by the JNI and DLL compilations. Please note that these settings are for the working environment not for the deployment. In the deployment, the environment is very simple. If we bundle the ORBacus with our classes in the same directory, we virtually need not set any environment variables provided that correct running environment for Java1.2 has been set up in the targeted machine.

Bibliography

[Adler95] R.M. Adler "Emerging Standards for Component Software" IEEE Computer 28 (3) 68-77 March (1995)

[Alenc99] P.Alencar and D.Cowan "The Role of Formal Method in Component Based Software Engineering" Proceedings of 1999 International Workshop on Component Based Software Engineering p193 May (1999)

[Becte90] J.C.M.Bacten and W.P.Weijland, "Process Algebra", Cambridge Tracts in Theoretical Computer Science 18, Cambridge Press, (1990)

[BenAr90] M. Ben-Ar "Principles of Concurrency and Distributed Programming" Prentice Hall (1990)

[Brown96] A.W. Brown (Ed.), "Component-Based Software Engineering", IEEE Computer Soc. Press, Los Alamitos CA, (1996)

[Brown98] A.W. Brown and K.C.Wallnan "The Current State of CBSE", IEEE Software Sept./Oct. (1998) pp.37-46

[Brown99] A. W. Brown "Building Systems from Pieces: Principles and Practices of Component-based Software Engineering." Submitted as a chapter for a CBD book in Sterling Software, (1999) (Private Communication)

[Bruns97] G. Bruns "Distributed systems with CCS" Prentice Hall (1997)

[Butle99] Butler Group "Component-Based Development--Application Delivery and Integration Using Componentised Software" http://www.butlergroup.com/cbdindex.htm

[Chung98] P.E. , S. Yajnik, D. Liang, J.C. Shih, C.Y. Wang and Y.M. Wang "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer" the C++ Report Jan 1998 also
http://www.bell-labs.com/~emerald/dcom_corba/Paper.html

[Coulo96] G. Coulouris, J. Dollimore and T. Kindberg "Distributed Systems Concepts and Design" Addison-Wesley (1996)

[DCE95] AES/Distributed Computing—Remote Procedure Call, Revision B, Open Software Foundation, http://www.opengroup.org/dce/index.html

[Dean97] J. C. Dean and M. R. Vigder "System Implementation Using Commercial Off-The-Shelf COTS Software" Proceedings of the Software Technology Conference (STC '97), Salt Lake City, Utah, 28 April-3 May, (1997) Also http://wwwsel.iit.nrc.ca/abstracts/NRC40173.abs

[DSouz98] D. F. D'Souza and A. C. Wills; "Objects, Components, And Frameworks with UML – the Catalysis Approach"; Addison-Wesley, Reading, Mass. (1998)

[Gokha96] A. Gokhale and D. C. Schmidt "Measuring the Performance of Communication Middleware on High-Speed Networks", Proceedings of SIGCOMM Conference, ACM 1996, Stanford University, August 28-30, (1996).

[Hazar98] S. Hazari "Evaluation and Selection of Web Course Management Tools" University of Maryland Report (1998) http://linus.umd.edu/webct/default.htm

[Hoarc85] C.A.R.Hoarc "Communication Sequential Process" Prentice Hall International (1985)

[IONA98] IONA Technologies: OrbixTalk 2.0 (1998) http://www.iona.com/products/messaging/talk/index.html

[Jacob97] I. Jacobson, M.Griss and P. Jonsson "Software reuse, Architecture, Process and Organization for Business Success" Addison-Wesley (1997)

[Kozac98] W. Kozacznski and G. Booch "Component-Based Software Engineering" IEEE Software Sept./Oct. (1998) p.34

[Ma98] Wei-hsiu Ma, Yen-Jen Lee, Davic H.C. Du "Video-Based Hypermedia for Education-on-Demand" Multimedia Jan-Mar, 1998 pp72-83.

[Meta98] Meta Group Consulting, "CORBA vs. DCOM: Solution for the Enterprise" March (1998) http://www.sun.com whitepapers/CORBA-vs DCOM.pdf

[Micro96] Microsoft "Distributed Component Object Protocol DCOM/1.0 Specification" May 1996, http://www.microsoft.com/oledev/olecom/dcomsepc.txt Together with also: the Microsoft COM and DCOM site http://www.microsoft.com/com/

[Milne89] R. Milner "Communication and Concurrency" Prentice Hall (1989)

[Marti99] D. Martin and J. Martin "Java and Digital Images" Dr. Dobb's Journal p.72 May (1999) and private communications

[Myer88] B. Myer "Object-oriented Software Construction". Prentice Hall (1988)

[OMG95] Object Management Group "Architecture and Specification" http://www.omg.org (1995)

[OMG98a] Object Management Group "CORBA/IIOP 2.2" http://www.omg.org/corba/corbaiiop.html (1998)

[OMG98b] Object Management Group "CORBA services Specification" http://www.omg.org/corba/sectran1.html (1998)
"CORBA facilities Architecture Specification" http://www.omg.org/corba/cf2.html (1998)

[OOC98] OOC: Source Code, Event Channel 3.1.1 Nov. 27th, (1998)
http://www.ooc.com

[OOC99a] OOC: ORBacus for C++ and Java, Version 3.1.3 (1999) http://www.ooc.com

[OOC99b] OOC: Source Code, Event Channel 3.1.3 Apr. 28th, (1999)
http://www.ooc.com

[Orfal96] R. Orfali, D. Harkey and J. Edwards "The Essential Distributed Object Survival Guide", John Wiley Press (1996)

[Schmi97] D.C. Schmit and S. Vinoski "Object Interconnections: Event Service" SIGS C++ report Magazine Feb, (1997)

[Schmi97a] D. Schmdit and S. Vinoski "Overcoming Drawbacks in the OMG Event Services" C++ Report Magazine June (1997)

[Schmi97b] D. Schmdit and S. Vinoski "An Introduction to CORBA Messaging" C++ Report Magazine June (1997)

[Schne94] J. Schnepf et al. "Closing the Gap in Distance Learning: Computer-Supported, Participate, Media-Rich Education", *Education Tech. Review,* No. 3. Autumn/Winter, (1994) pp19-25.

[Siegel96] J. Siegel *et.al.* "CORBA Fundamentals and Programming" Jon Wiley &Sons (1996)

[Silbe98] A. Silberschatz and P.B. Galvin "Operating System Concepts" Addison-Wesley (1998)

[Sun97] Sun Microsystem Inc. Java RMI Specification, Internet Draft http://www.javasoft.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html Dec. (1997)

[Sun98] Sun Microsystem Inc. Enterprise JavaBean Specification 1.0 (1.1) http://java.sun.com/products/ejb/docs.html Mar (1998)

[Szype98] C.Szyperski "Component Software: Beyond Object-Oriented Programming", Addison-Wesley (1998)

[Tjand99] I.A. Tjandra and J.X. Wang "Distributed System Development with the Assembly Line Pattern", Parallel and Distributed Processing Techniques and Applications (PDPTA'99) (1999)

[Vinos98] S. Vinoski "New Features for CORBA 3.0" Communications of the ACM, Vol 41 pp 44-52 Oct (1998) For more about "CORBA 3" see also http://www.omg.org/news/pr98/component.html

[Vogel97] A. Vogel and K. Duddy "Java Programming with CORBA" Jon Wiley & Sons (1997)

Vita Auctoris

Name:	Yongliang Shao
Place of Birth:	Tianjin, China
Year of Birth:	1962
Education:	Tsinghua University, Beijing, China 1980-1985 B.Sc. in Eng. Physics
	Institute of Electronic Structure and Laser University of Crete, Crete, Greece 1988-1993 Ph.D. in Physics
	University of Windsor, Windsor, Ontario, Canada 1998-2000 M.Sc. in Computer Science