♦ Communication Middleware for Multi-Party Multimedia Applications

J. Robert Ensor and Sudhir R. Ahuja

This paper describes a communication middleware system called CoMMware. The CoMMware software package contains a set of modules that supports a wide variety of multimedia communication applications, such as real-time conference programs, distance learning tools, and distributed games. It encourages modular program development and software use through its registry and brokering mechanisms. CoMMware also supports behavior policies, which act as plug-ins for controlling how program modules interact with each other. In addition to aiding development of distributed programs, communication middleware provides a means of exchanging data in multiple media. CoMMware's virtual transport is a programming interface for specifying communication paths among program modules and qualities of service for transmission of multimedia data over these paths. CoMMware also helps people use multimedia applications. The system's persistent environment helps users locate and access applications during real-time interactive communication sessions, and it coordinates information storage between these sessions. The paper also presents brief overviews of some applications built with CoMMware.

Introduction

CoMMware is a communication middleware system. Using a common overall architecture, distinct versions have been built in Bell Labs and in Lucent Technologies' Business Communications Systems Division. This paper describes the Bell Labs version.

CoMMware is a collection of software modules that helps developers write multi-party multimedia applications, such as real-time conference programs, distance learning tools, and distributed games. As **Figure 1** illustrates, this collection of software components may be characterized as a conceptual layer, situated *between* higher-level application programs and underlying computation and communication systems. Middleware, which hides details of lower-level operating systems and networks from the application developer, is described by and made available through its application program interfaces. Thus, middleware creates a stable programming base for building applications that can execute on various computer systems and communication networks.

While the more familiar and fundamental middleware provides distributed programming tools, communication middleware also provides quality-of-service communication support. Applications can use this program base for an important set of core functions including multi-party session management and multimedia connection management. Furthermore, these core functions are available on several operating system and network platforms. Therefore, an application—for example, an audio-video conferencing system—could be built using CoMMware to execute on both local area network (LAN) and integrated services digital network (ISDN) platforms.

CoMMware helps people build distributed programs and encourages construction of applications through composition of program modules. Its program development environment includes a definition lan-

Panel 1. Abbreviations, Acronyms, and Terms

ACM	Association for Computing Machinery
CORBA	Common Object Request Broker Architecture
ISDN	integrated services digital network
ITU-T	International Telecommunication Union – Telecommunication Standardization Bureau
LAN	local area network
MMCX	MultiMedia Communications eXchange
MSS	Multimedia System Services
OLE/COM	object linking and embedding/component object model
SIGCHI	Special Interest Group on Computer-Human Interaction
TINA-C	Telecommunications Information Network Architecture Consortium

guage for module interfaces and a registration mechanism that makes these program modules available for use in applications. The CoMMware run-time environment is a set of processes and application library elements that permits application modules to interact. When an application module invokes a registered program module, CoMMware *broker components* query a CoMMware-maintained database to locate the called module. The broker components then manage execution of the called module. Data is exchanged between the program parts through the calling module's parameters. CoMMware also provides a means of defining and using *behavior policies*, which are software plug-ins for controlling how program modules interact.

CoMMware supports communications among application program modules by providing:

- Mechanisms for signaling among modules,
- Control for connections that are used by application modules to exchange data (in one or more media), and
- Communication session control.

All these services use underlying transport primitives. However, they are built independently and can be realized using distinct networks. Hence, CoMMware is suitable for application development in several network environments. CoMMware has been used to create applications that execute on LANs, private branch exchanges, ISDN, and asynchronous transfer mode broadband networks.

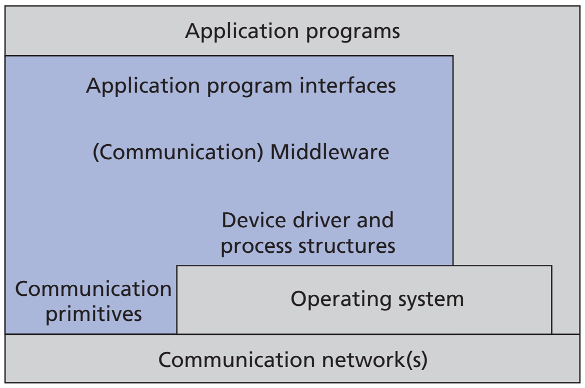
Application programs
Application program interfaces
(Communication) Middleware
Device driver and process structures
Communication primitives
Operating system
Communication network(s)

Figure 1.
Conceptual view of middleware.

The CoMMware run-time environment manages groups of modules. It creates transaction mechanisms to define atomic operations on these groups. The transactional group operations include support for signaling among modules located within a group. The operations provide event distribution among group members. This mechanism supports synchronization among program elements. The group operations also provide program developers with a composition mechanism that permits dynamic run-time construction of applications.

CoMMware provides *virtual transport*, which acts as an interface to underlying communication networks. The virtual transport operations establish and manage media connections among program elements. They also include a means of specifying the required qualities of service—that is, performance characteristics for multimedia communications. The virtual transport component also helps developers specify distribution paths for information, which aids the building of multi-party applications.

CoMMware provides a multi-party session control mechanism called a *context*. Traditional call management functions are supported by contexts. Because CoMMware contexts can persist longer than a single communication session, they also provide a mechanism for conducting a series of related communication sessions. People and programs can rendezvous for a communication session using a CoMMware context. Context persistence is also a means of supporting mobile access to services. That is, a user can access a context while moving among several locations (and changing input/output devices).

The CoMMware execution environment also supports users of applications. For example, suppose Kate is the user of a multimedia conferencing system that has been constructed with CoMMware. She could access the application within a context by issuing commands from her office computer. CoMMware is responsible for providing Kate with a session and for establishing signaling and transport connections among the application components. If the conferencing application requires bridging services during the conference, they can be added to the context using CoMMware group operations. Kate may then leave her office and drive home. During the drive, she may re-enter the context, accessing the application (and perhaps talking with other parties) through her car telephone. At her house, she can use a set-top box to re-establish contact with the CoMMware server and re-enter the context.

Background and Related Work

Most familiar instances of middleware—for example, the Object Management Group's Common Object Request Broker Architecture (CORBA)[1] and Microsoft's object linking and embedding/component object model (OLE/COM)[2]—are targeted for the support of distributed programs. These systems aid program creation and description by defining an interface description language that allows program elements to be described independently of the programming language used in their implementation or invocation. These middleware systems also specify mechanisms that allow program elements to be registered with a name service. When one program element requires service from a second element, it sends a request to a component of the middleware execution environment. This component is called the *broker*, and it selects one of the registered program elements to supply the requested service. The program elements are then able to communicate using information exchange standards defined by the middleware. The standardized description of programs and their input/output makes invoking and controlling services easier while supporting reuse of existing programs.

The Multimedia System Services (MSS) proposal within the Interactive Multimedia Association[3] and the Telecommunications Information Network Architecture Consortium (TINA-C) proposal[4] represent a specialized class of communication middleware. Both MSS and TINA-C are based on the CORBA specifications. However, these proposals not only support transmission of requests and responses among program elements. They also support exchange of time-dependent data (for example, voice and video) in multimedia applications. These proposals define the means of specifying quality-of-service parameters and synchronization among multiple information streams. While the first class of middleware is useful over a wide range of data communication networks, the second class depends more on the performance offered by underlying multimedia networks. CoMMware is also communication middleware. One version of CoMMware has been built with OLE and another with a CORBA-like framework. (The CoMMware virtual transport is a predecessor of the proposed WinSock 2.0 open-network Application Program Interface (API) standard, which could be used for this CoMMware interface.)

Communication middleware proposals represent services that closely resemble those offered by advanced signaling protocols. For example, the International Telecommunications Union H-series recommendations[5] specify quality-of-service-based communications on a suite of networks. However, the H-series protocols bundle session control, media connection support, and coding schemes with application management. These protocols do not permit platform-independent application development. For example, H.320 applications are dependent on ISDN-based session control, connection management, and coding schemes. On the other hand, middleware need not be restricted to any one underlying network. Therefore, middleware can offer a common application programming interface for a variety of underlying networks (and associated signaling protocols). This transport independence is a major strength of communication middleware.

CoMMware goes beyond other communication

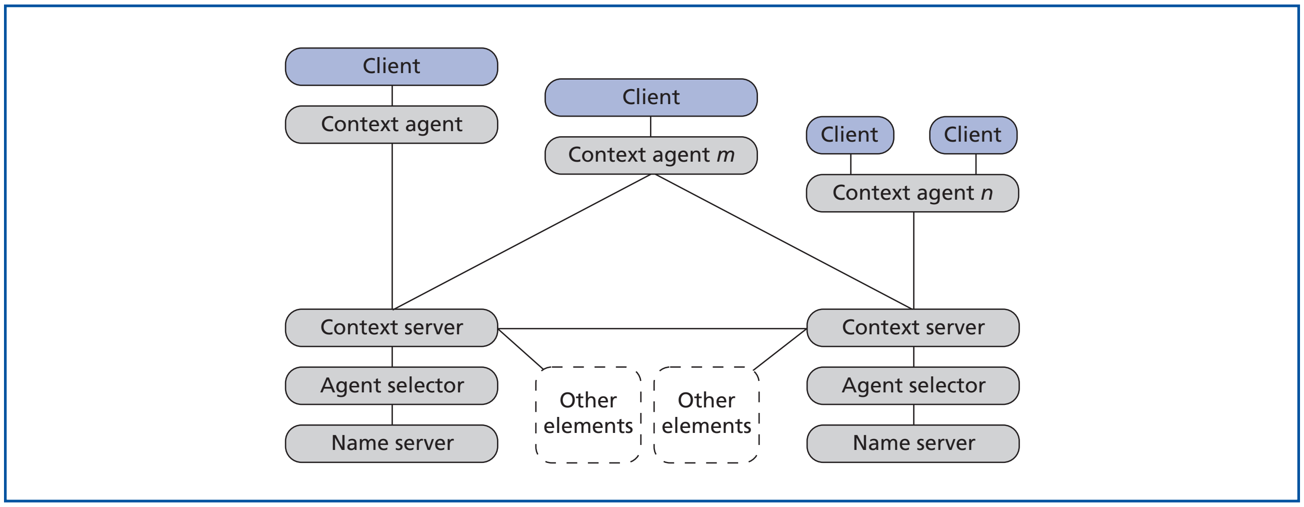

Figure 2.
CoMMware architectural elements.

middleware and signaling proposals. It provides support for *collections* of program modules, applications, and communication sessions. The behavior policies of CoMMware provide a unique means of controlling collections of program modules. The CoMMware context provides mechanisms for users to control multiple applications. The context also provides a stable environment within which to control multiple sessions. Only CoMMware creates a persist communication environment that relates applications and multiple communication sessions. In conventional communication systems, any higher-level context extending beyond a single session (which might exist among the communicating parties) must be maintained by those parties. For example, no communication standard helps conference participants store meeting minutes on a file server.

System Architecture

CoMMware is built as a specialized client-server system. A *context agent*, which is a client of a CoMMware context server, represents each user. A *context server* provides two sets of resources to its clients—*contexts* and *sessions*. A context (or call environment) is a persistent electronic environment within which users may access collections of communication-based applications. Once a context has been created, its description is maintained by a context server until a user (with the necessary authority) requests that the context be destroyed. Thus, a context serves as an electronic place of rendezvous—that is, a meeting place—and also as an electronic environment within which to create and offer services—that is, a place for information storage and exchange. A *session* (also termed a *call* or a *conference*) is a period of user activity within a context.

Figure 2 illustrates the fundamental elements of the CoMMware architecture—context servers and context agents—as well as clients of the CoMMware system. These clients are called context service clients (or *derived* applications). The term "derived" is used because the applications derive part of their functional qualities from CoMMware. (The term does not refer to derived classes within object-oriented programming languages.) Context servers and context service clients are described as separate loci of activity. Each context service client is connected to a context agent so that it can communicate with context servers by means of a proprietary protocol. A context service client can provide a local representation of a context for users or other application programs.

The figure also explicitly represents two other CoMMware elements, which support its two fundamental elements: the *agent selector* and the *name server*. Additionally, unspecified supporting elements, such as security and billing modules, are indicated as a collection and labeled as "other elements" in the figure.

Figure 2 also shows the communication connections among the above-mentioned elements. Each communication connection may be produced by the CoMMware virtual transport feature. Virtual transport is a software module that abstracts details of underlying physical communication networks to provide a stable application program interface for a variety of communication networks. Many physical realizations of these connection abstractions are possible. These realizations may vary from shared memory structures within a single computer to collections of channels within a collection of wide area networks.

Context Server

A CoMMware context is an object (that is, a software data structure and associated procedures) representing the abstract notion of a communication context. Many implementations of the context object are possible. For the purposes of exposition, this paper describes a context object's representation in terms of the context server introduced above.

For simplicity, we elaborate our description of the representation in terms of two restrictions. First, we avoid descriptions of distributed consistency controls by assuming that each context object is managed (created, manipulated, and destroyed) by exactly one context server. Second, we assume that a context server may represent more than one context object at a time.

The context abstraction is defined in terms of a state machine and the messages that trigger its transition from one state to another. Correspondingly, we describe a context server as maintaining a finite-state machine for each context that it manages. Therefore, each context server contains a component known as the *engine* that acts as a manager of its finite-state machines. The messages triggering context state transitions are received by the state machine's managing context server and may come either from context service clients (via context agents) or other context servers. As a resource manager, a context server allocates identifiers for its managed resources. Context servers allocate context identifiers in a way that guarantees each identifier is globally unique.

Context Agent

The conduit between context servers and context service clients is called a *context agent*. While the actual participants in CoMMware protocol exchanges are context servers and context server clients, the protocol messages are transmitted between these parties through context agents.

A context agent is a "port" for a set of clients to one or more context servers. A context agent is also a port for a context server to one or more of its clients. With this communication indirection, CoMMware can provide context services, such as authentication and billing, independently of any other specialized services that might be accessed within a context. The indirection also supports the multiplexed handling of protocol messages on server and client computers. It also permits flexible techniques for building dispatching policies for the context messages that are associated with a given single context.

A context agent serves as the dispatcher for events to and from one or more context servers simultaneously for communication sessions *involving different contexts*. An example of this situation is illustrated by the context agent *m* in Figure 2, which is interacting simultaneously with two context servers. Similarly, a context agent can interact with more than one context service client simultaneously for communication sessions *involving different contexts*. Figure 2 illustrates this situation as well. Context agent *n* interacts with multiple clients simultaneously. Furthermore, a context agent can dispatch events from concurrent activities within a single context—that is, it can act as an *dispatcher*.

However, the distribution of events has fundamental impact on service interactions when multiple applications are active in a single context. Therefore, in some cases, the event dispatch policies should become part of a user's view of the application environment. This need is easily met in the CoMMware architecture. One simply makes a context service client the dispatcher for the service-specific messages that go to and from all applications in a given context. When serving in this role, the client is termed an intracontext dispatcher. This dispatch structure permits application-level control of interactions among services—that is, all policies regarding service interaction are implemented explicitly in application space.

Name Server and Agent Selector

This paper does not contain detailed behavioral

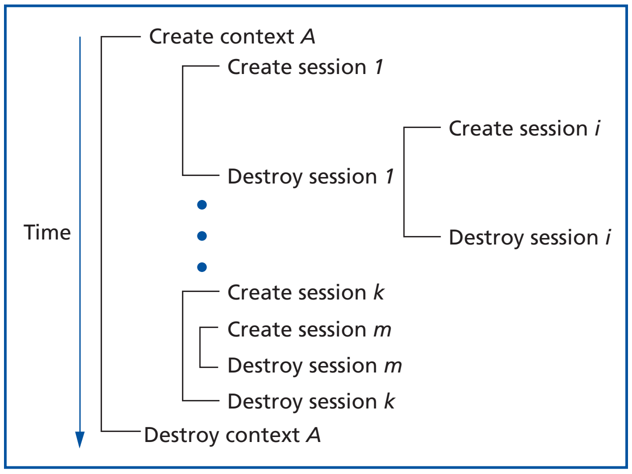

Figure 3.
The scope of context and sessions.

specifications for the other CoMMware architectural elements. The elements that support the components described above have little impact on the protocol exchanged between context servers and their clients. However, we briefly mention here the roles played by the supporting elements in CoMMware service, and we note some of the information needed by the supporting elements to effect these roles.

CoMMware architectural elements communicate with one another. Therefore, all elements that are distinct implementation units must be named to establish communication paths. Because context servers provide uniquely identified context resources for their users, the context servers themselves have globally unique names, which permit rendezvous and access to shared resources. The context servers interact with their clients through context agents. Therefore, the context agents also have globally unique names. One or more name servers contain the communication network ports as values corresponding to these names. Context service client names are local to their corresponding context agents, so their names may be maintained within context agent name spaces.

In the CoMMware architectural model, a context server does not access a name server directly to resolve a context agent name. Rather, it interacts with an agent selector, which, in turn, uses one or more name servers. This interaction occurs when a context server responds to a service request. The selection of an appropriate agent may be effected in a variety of ways. For example, selection may be accomplished either through look-up in a static table or through evaluation of a set of dynamic state descriptors, including the current operation field(s) and the current states of the context server engine, selector, and a context service client. In the latter example, the selector must communicate with other architectural elements to gain the necessary state information.

Other Supporting Elements

Authentication and billing components and other supporting elements are important parts of CoMMware service. However, these elements play a secondary role in the protocols described below. Their impact is limited to the interpretation steps performed by the context server. Hence, their impact on the protocol can be limited to changes in the information that is included in the parameters of requests to context servers and of responses from the context servers.

Virtual Transport

CoMMware is a distributed software system. Therefore, components of CoMMware must be able to communicate with one another. To support such communication, CoMMware contains an independent software module called *virtual transport*. CoMMware's virtual transport provides a means for context servers to communicate with context agents and other context servers.

The CoMMware model assumes that protocol messages among context servers and context agents are exchanged over reliable transport. The virtual transport connections between context servers and context agents may also be used as a means of exchanging negotiation messages among context agents (on behalf of context service clients) for end-to-end application-specific negotiations. In addition, virtual transport functions are available for use outside the CoMMware context framework. They may be used for application-specific payload messages.

Session Control

CoMMware creates an execution environment that explicitly represents communication contexts as persistent data records and uses this information to structure user communications and access to sets of shared data. **Figure 3** illustrates the relationship between contexts and periods of activity within these

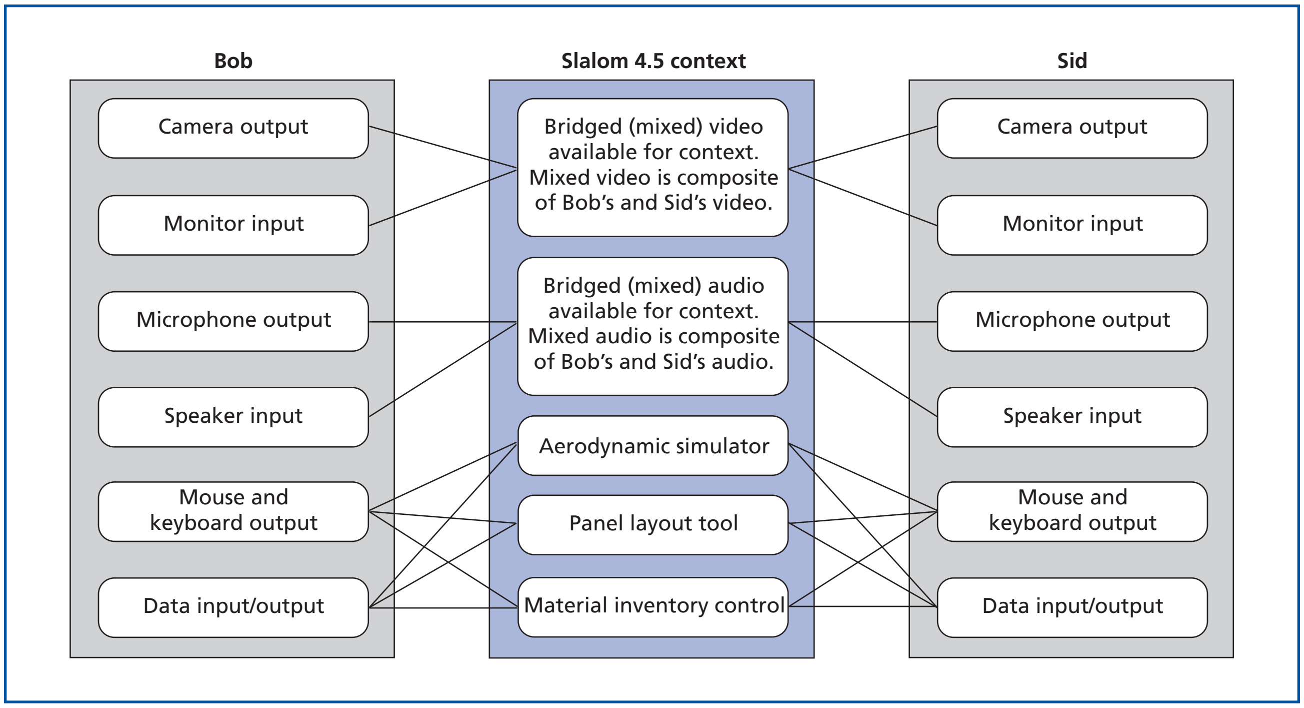
Bob
Slalom 4.5 context
Sid
Camera output
Monitor input
Microphone output
Speaker input
Mouse and keyboard output
Data input/output
Bridged (mixed) video available for context. Mixed video is composite of Bob's and Sid's video.
Bridged (mixed) audio available for context. Mixed audio is composite of Bob's and Sid's audio.
Aerodynamic simulator
Panel layout tool
Material inventory control
Camera output
Monitor input
Microphone output
Speaker input
Mouse and keyboard output
Data input/output

Figure 4.
Typical use of CoMMware context.

contexts, which are labeled *sessions*. A session must take place within a context; hence, the scope (lifetime) of a session must be contained within the scope of a context.

A session can take place either in a newly created context or in some existing context. Thus, traditional session management can be effected within CoMMware by creating a context when the session begins and destroying the context when the session ends. This approach restricts the scope of the context to equal the scope of a session. However, CoMMware treats contexts and sessions more generally than conventional session managers.

A CoMMware context can be created before a session is held within it and it can exist after the session has ended. Thus, the context is an environment that can relate sessions. A session can be started in an existing context, which might contain state that is serving as higher-level context for each session participant. This is a means of relating sessions held sequentially in the same context. Concurrent sessions can also be related by sharing a context. For example, while using a collection of applications in one session, a user might access another set in a second session. In addition, CoMMware permits a session to have only one participant. Consequently, a person or program can use a session for individual purposes or as a rendezvous mechanism—a means of waiting for another party.

Through trials with CoMMware-based systems, we have seen that persistent contexts are especially useful for long-term collaborations. **Figure 4** represents an example of the use of a CoMMware context in a collaborative situation. The scenario illustrated is one in which Bob and Sid are long-term collaborators on the design of a windsurfing sail. They have used CoMMware to create a long-lived environment called *Slalom 4.5* for their project. This context is used as their meeting place and as a repository for their shared information regarding the project.

During a typical session, Bob and Sid use a collection of derived applications to support their conversations—namely, audio, video, and program sharing applications. To help with their design tasks, the designers also make use of some specialized CAD tools—an aerodynamic simulator, a panel layout tool, and a material inventory control program. During the session, each derived application might require multiple connections from the underlying multimedia net-

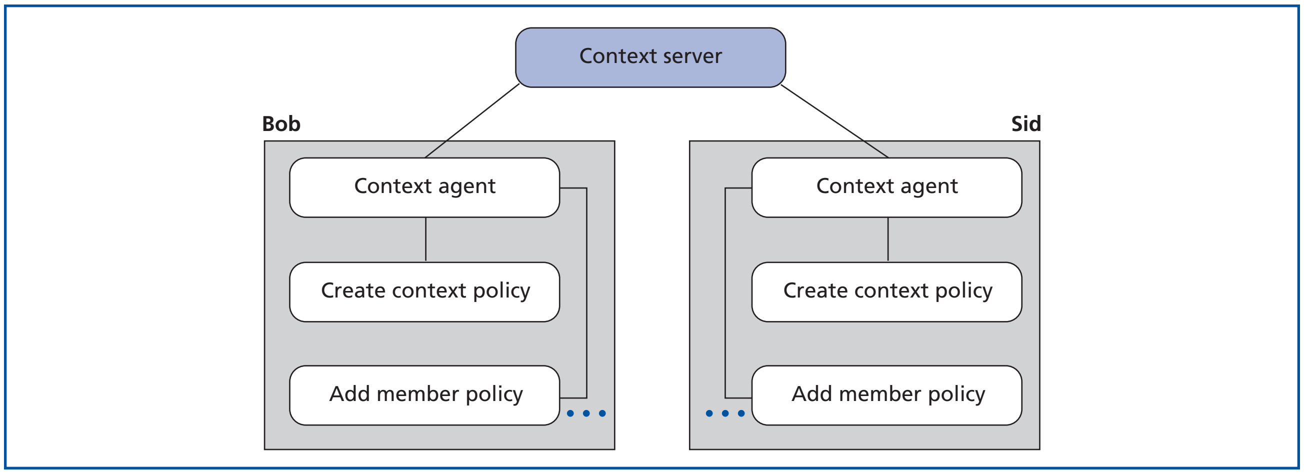

Figure 5.
An example of negotiation and policies.

work to move the information to and from Bob and Sid. After a design session has ended, the multimedia network connections can be released. Later, Bob or Sid can use CoMMware to re-enter the Slalom 4.5 context, re-establish network connections to the appropriate applications, and continue work. A subsequent session can continue the design in the same environment.

Negotiation and Policies

The CoMMware context operations per se are limited to a set of group management operations. In this respect, the CoMMware context shares properties with other group management systems—for example, the Isis system[6]. Some common characteristics, such as atomicity of operations, are addressed in conventional ways by CoMMware. The primary operations defined for contexts are also those associated with typical group management—that is, create and destroy the group and add and drop members to and from the group. Membership in a context is limited to context service clients. As discussed above, each context service client interacts with context servers through context agents.

In general, group members are geographically distributed. Hence, CoMMware must establish and manage communications for group members and potential group members. Communications are needed for a user to make requests to CoMMware and for CoMMware to respond to a user. CoMMware also sends information to all group members when their group state changes. In addition, CoMMware offers communication services to group members so that distribution of information among these sets of users can be precisely controlled. The message distribution services provide various ordering and reliability properties.

Group members may be involved in operations that affect their group. Whenever CoMMware receives a request to alter the state of a group, it offers the group members a chance to accept or reject the proposal or to exchange information with each other regarding the proposal. Each member can define its own policies regarding an operation. Furthermore, if the member has the necessary authority, it can exercise these local policies during group operations. For example, as members of the Slalom 4.5 group, the two parties named in **Figure 5** (Bob and Sid) can influence the handling of operations on that group. In one scenario, Bob proposes to add a video server to the group. Sid, however, thinks that this medium represents a costly and unnecessary service, and he can execute a policy that rejects the proposal.

Event Distribution

Acting on behalf of context service clients, a context server receives messages from context agents and other context servers. In response to these messages, the context server may distribute events to context agents (for context service clients) and other context servers. As the context server interprets the message, it manages distribution of events generated by this interpretation.

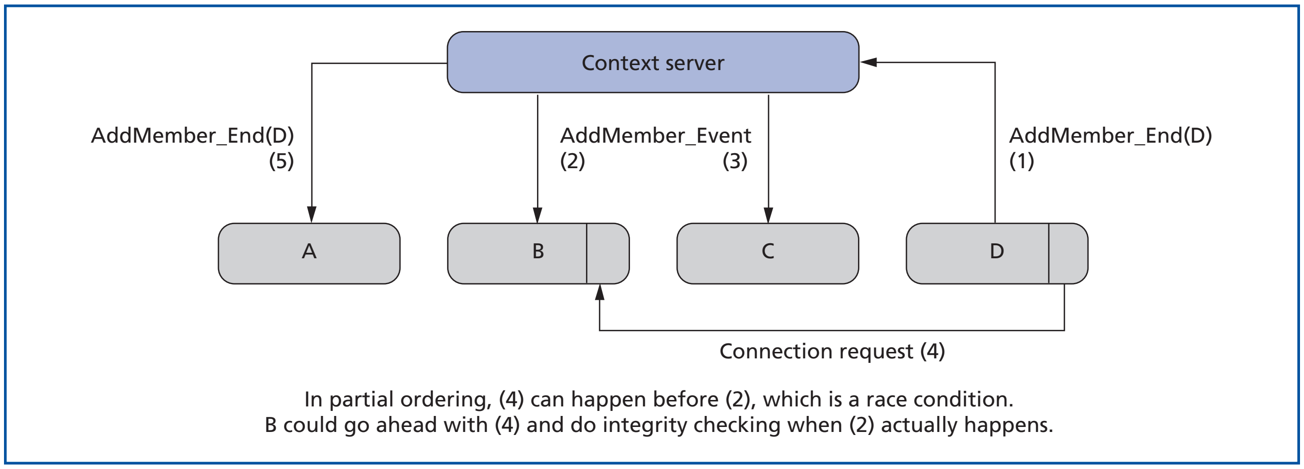
Context server
AddMember_End(D) (5)
AddMember_Event (2) (3)
AddMember_End(D) (1)
A
B
C
D
Connection request (4)
In partial ordering, (4) can happen before (2), which is a race condition.
B could go ahead with (4) and do integrity checking when (2) actually happens.

Figure 6.
Partial ordering of events.

Event Ordering and Race Conditions

When a context server interprets a context server or context client protocol message, it may send events to several context service clients through their context agents. The event distribution by a context server is effected under one of two event ordering schemes: *partial ordering* or *total ordering*. Partial ordering generates less message overhead and permits clients to react more quickly to events. However, race conditions, or events executed out of sequence, can arise from *out-of-band communication* among clients following partially ordered event distribution within a transaction. Total ordering of event distribution generates more message exchanges and delays agent reaction to events. However, the total ordering mechanism helps prevent race conditions among clients.

Two examples illustrate the uses of partial and total ordering of event distribution. Suppose context server *S* manages context *X* with members (context service clients) *A*, *B*, and *C*. Suppose further that member *A* adds a new member *D* to the context *X*. The *AddMember* transaction begins with the `AddMember_Begin` request, continues with some negotiation messages, and ends with an `AddMember_End` request. In the first example, which **Figure 6** illustrates, context server *S* uses the partial ordering scheme to distribute events generated during its interpretation of the *AddMember* transaction. In the second example, which **Figure 7** illustrates, *S* uses the total ordering scheme to distribute the same events.

Suppose that after the `AddMember_Begin` and negotiation messages have been handled by the context server *S*, it receives an `AddMember_End` request from *D* (shown as message 1 in Figures 6 and 7). Interpreting this request, *S* broadcasts the `AddMember_Event` to the other members of the context (*B* and *C*) indicating that *D* has been added to *X*. Meanwhile, *D*, now acting as a member in the context, establishes an out-of-band connection with *B*. (The user represented by *D* might have attempted to connect to a server associated with *B*, for example). B might not have received the `AddMember_Event` from the context server yet and, hence, may not know if it should honor the connection request. As a policy, *B* might accept the connection and defer the integrity checking until it receives the `AddMember_Event` or a time out. Such a policy would permit *B* to offer service quickly but with some danger of security breaches.

Let us now consider the total ordering of events as Figure 7 shows. When the new member *D* sends the `AddMember_End` request, the transaction still is not committed by the context server. No member can assume that the transaction is committed until it receives a transaction commit event from the context server for this transaction. The context server sends the `AddMember_Event` to all the other members of the context and waits for the acknowledgment (note the existence of a time-out policy here in the context server for the acknowledgment wait).

After it receives the acknowledgment from all the

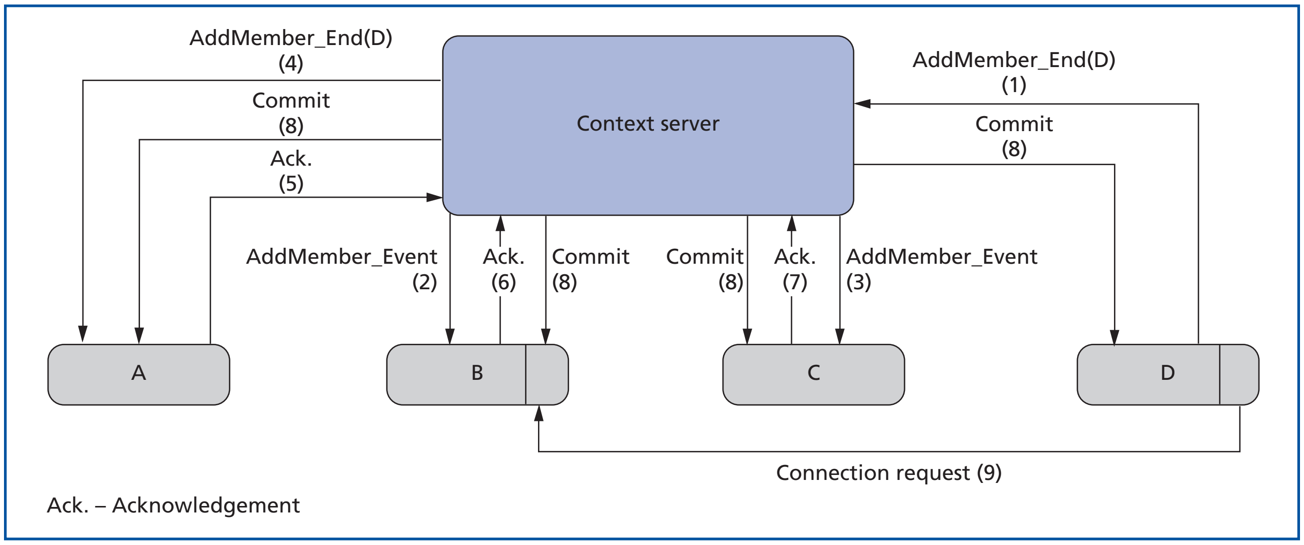

Figure 7.
Total ordering of events.

context members, the context server sends a transaction commit event to all the members, including the newly added member. Any member can assume that the transaction has committed only at this point. Note that the context server need not wait for the acknowledgment of the transaction commit event. Now, if *D* attempts an out-of-band communication with *B* based on the previous transaction, *B* will honor the request appropriately.

Experiences

CoMMware has been used as a middleware component—that is, a foundation of program modules—in the construction of several multimedia applications. Two applications, Visual Meeting Minutes[7] and Archways[8], were discussed in a previous paper.[9] Recently, the Persyst[10] and MultiMedia Communications eXchange (MMCX)[11] applications have been built with CoMMware.

Persyst is a software system that helps support distance learning by providing teachers and students with virtual classrooms. This system creates a collection of virtual rooms called a *course*, which corresponds to an academic course. The teachers and students of an academic course interact in the corresponding Persyst course, accessing its virtual rooms through the Internet or an intranet. Each of these virtual rooms provides specialized functions related to teaching and learning. For example, Persyst defines rooms for course information, libraries, recitation halls, lecture halls, and calendars.

Persyst is built as an application on the Bell Labs version of CoMMware. Each Persyst course and classroom is based on a CoMMware context. Context service clients representing teachers and students are members of contexts associated with a course. Any communication servers, such as audio and video bridges, which support interactions among teachers and students during a session within a virtual room, are also members (through their program representations) of the corresponding contexts. Course data, including calendars and lecture notes, are stored by Persyst servers, which are also members of course and classroom contexts. Thus, Persyst users interact with each other and access course data in virtual rooms.

MMCX is a system that supports collaborative work. It provides its users with a multimedia conferencing system featuring real-time interaction among several parties through voice, video, and data communications. MMCX also provides such calling features as call coverage and call forwarding for both multimedia and conventional voice-only calls. MMCX is built on Lucent Technologies' Business Communications Systems version of CoMMware. In addition, CoMMware contexts are used as the basis for MMCX call management operations and for the coordination of media servers during multimedia calls. Specific behaviors are achieved through policies invoked during negotiations.

Summary

CoMMware is a communication middleware package supporting development of multimedia applications, such as real-time conference programs, distance learning tools, and distributed games. It encourages modular program development and software reuse through registry and brokering mechanisms. It also provides a framework within which behavior policies can act as plug-ins for controlling how program modules interact with each other.

CoMMware also supports communication among program modules. It provides means of signaling among the program elements through partial and total ordering of events. Its virtual transport interface provides a means of controlling real-time communication of multimedia information among program modules. This interface allows programmers to specify both communication paths among program modules and the performance characteristics of these paths.

CoMMware's execution environment, based on contexts, also helps people use these multi-party and multimedia applications. Its persistent environment helps users locate each other and establish communication sessions. It also helps them access applications during these sessions, and it coordinates information storage between sessions.

CoMMware has proven useful in building several multimedia communication applications. It has served as a middleware code base for these systems, making them useful in a variety of computing and communication environments. CoMMware-based applications are currently used with different operating systems, as well as LAN, Internet, and various telephony networks.

Acknowledgments

Murali Aravamudan played a central role in CoMMware's development. He was one of its principal architects, leading implementation of the Bell Labs version and helping to write system descriptions. Many colleagues (too numerous to list) in Lucent's Business Communications Systems Business Unit helped define the system's architecture and developed a version as part of the MMCX product. The authors gratefully acknowledge the work of all these individuals.

References

1. Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, John Wiley, New York, July 1995.
2. *OLE 2 Programmer's Reference*, Volumes 1 and 2, Microsoft Press, Redmond, Washington, 1994.
3. *Multimedia System Services, Project Description*, Interactive Multimedia Association, Annapolis, Maryland, June 1993.
4. T. Handegard, "TINA-C Computational Modeling Concepts," Telecommunications Information Network Architecture Consortium (TINA-C) Document TP_HC.012_3.2_96, Red Bank, New Jersey, May 17, 1996.
5. *Line Transmission of Non-Telephone Signals*, ITU-T Series-H Recommendations H.100 through H.331, International Telecommunications Union – Telecommunication Standardization Bureau (ITU-T), Geneva, Switzerland. http://www.itu.ch/itudoc/itu-t/rec/h.html
6. K. P. Birman, "The Process Group Approach to Reliable Distributed Systems," *Communications of the Association for Computing Machinery (ACM)*, Vol. 36, No. 12, Dec. 1993, pp. 36-53.
7. A. B. Ginsberg and S. R. Ahuja, "Automating Envisionment of Virtual Meeting Room Histories," *Proceedings of the Association for Computing Machinery (ACM) Multimedia 95*, San Francisco, California, Nov. 1995, pp. 65-75.
8. D. D. Seligmann, R. T. Mercuri, and J. T. Edmark, "Providing Assurances in a Multimedia Environment," *Proceedings of the Association for Computing Machinery, Special Interest Group on Computer-Human Interaction (ACM SIGCHI)*, Denver, Colorado, May 1995, pp. 250-256.
9. D. A. Berkley and J. R. Ensor, "Multimedia Research Platforms," *AT&T Technical Journal*, Vol. 74, No. 5, Sept./Oct. 1995, pp. 34-45.
10. *Welcome to Persyst* (general overview of the Persyst virtual classroom), Bell Labs Multimedia Communication Research Department, Holmdel, New Jersey, 1997. http://www.multimedia.bell-labs.com/projects/persyst
11. B. S. Katz and L. M. Sanders, "MMCX Server Delivers Multimedia Here and Now," *AT&T Technology*, Vol. 10, No. 4, Winter 1995-96, pp. 2-6.

(Manuscript approved March 1997)

J. ROBERT ENSOR is a distinguished member of technical staff in the Multimedia Communication Research Department at Bell Labs in Holmdel, New Jersey. He conducts research in multimedia communication systems and currently specializes in Internet-based applications. Mr. Ensor holds a B.S. degree in biology from Furman University in Greenville, South Carolina, as well as M.S. and Ph.D. degrees in computer science from the State University of New York in Stony Brook.

SUDHIR R. AHUJA is head of the Multimedia Communication Research Department at Bell Labs in Holmdel, New Jersey. He is responsible for research in multimedia collaboration, communication, applications, and platforms, as well as research and technology transfer of key services, such as multimedia conferencing and multipoint audio and video bridging. Mr. Ahuja received a B.Tech. degree in electrical engineering from the Indian Institute of Technology in Bombay, as well as M.S. and Ph.D. degrees in computer science and electrical engineering from Rice University in Houston, Texas. ◆