Caltech Infospheres Project Overview: Information Infrastructures for Task Forces

K. Mani Chandy*
Computer Science 256-80
California Institute of Technology
Pasadena, California 91125
mani@cs.caltech.edu
http://www.infospheres.caltech.edu/

November 14, 1996

The organization of this document is as follows: (i) the problem domain, (ii) research we are planning on doing, (iii) the context of the planned research in relation to work done elsewhere, (iv) tasks and deliverables, and (v) research completed on active object networks.

1 Problem Domain and Relevance

1.1 Vision

The vision that drives the research described in this document has two components: (i) increasing use of fluid adaptive organizational structures and task forces, and (ii) developments in network-centric technologies. This document discusses both aspects of this vision, and then defines a problem: the design of information infrastructures that exploit network-centric technologies and support task forces. We analyze the requirements of such information infrastructures, and propose a preliminary design that satisfies the requirements.

1.1.1 Fluid, Adaptive Organizational Structures and Task Forces

Organizations of all types — in government, industry and academia — are changing. Traditional organizations with fixed sets of people and resources, carrying out fixed sets of activities, are giving way to fluid organizations that link different groups of people and resources temporarily to carry out missions as they arise. We use the phrase *virtual organization* (an extension of virtual corporation), for such fluid adaptive entities. Next, we give a few examples of virtual organizations.

Crisis Management. A task force that handles a crisis, such as the crash of an airplane, is a virtual organization. The task force draws members from different organizations such as the FBI, National

*The Infospheres Project is sponsored by the Air Force Office of Scientific Research under grant AFOSR 91-0070, by the CISE directorate of the National Science Foundation under Problem Solving Environments grant CCR-9527130, by the Center for Research in Parallel Computing under grant NSF CCR-9120008, by the Advanced Research Projects Agency, and by Novell, Inc.

Transportation Safety Board, the Coast Guard, the county coroner's office, state and local police, local hospitals, and civilian organizations such as the Red Cross. The task force has its own management and information structure that coexists with those of the organizations from which it draws its members. Predicting the next crisis is difficult, and so predicting the members of the next crisis-management task force is also difficult. Nevertheless, crisis-management teams, and the information infrastructure that supports them, must be set up very rapidly.

Proposal Review. Another example of a virtual organization is a panel that reviews proposals submitted to an agency, say AFOSR. AFOSR draws potential panel members from a large pool. When reviewing is complete, the panel is disbanded. The panel has its own organizational, information, and workflow structures that coexist with those of AFOSR. In essence, AFOSR expands its informational structures to include those of the panel, for the lifetime of the panel. In this sense, AFOSR's informational structures adapt in a dynamic, but systematic, way to include new people and resources as needed. Many institutions in government and industry respond to opportunities and crises by creating temporary, focused task forces composed from people and resources from within and from outside the institution. A key to the effectiveness of such task forces is the rapidity with which information infrastructures that match the task force's organization can be set up.

Research Consortia. A consortium of universities, that collaborate to do research, is another example of a virtual organization. Also, it draws its members from a large pool, has a limited lifetime, and has its own organizational, informational and workflow structures that coexist with the existing structures of the participating univerisities.

Health Delivery Systems. The duration of the relationships between groups of patients, doctors, psychologists, pharmacies, medical laboratories, and insurance companies, will decrease as health-care services become more generic and competitive pressures increase. The information infrastructure of health-delivery systems must span medical monitoring devices and other appliances such as computers in patients' homes, cars, and offices. The infrastructure must also link the providers of different medical services as they enter the system, and delete the links when they leave.

1.1.2 The Problem

Our problem is designing information infrastructures that support virtual organizations. The infrastructure should exploit the continuing rapid advances in network-centric tools, and tailor the tools to meet the needs of virtual organizations. An example of a virtual organization is a task force. Next, we discuss developments in network-centric technologies that can be used to develop a worldwide active object network that supports the information infrastructures of large numbers of evolving virtual organizations.

1.1.3 Developments in Network-Centric Technologies

The global information infrastructure will span billions of active persistent communicating objects. Just as you can set up a conference call connecting any arbitrary group of telephones from a pool of millions of telephones worldwide, an active object will be able to set up a collaborative session between an arbitrary group of objects from a pool of billions of objects worldwide.

Many of us will have objects that manage our documents, calendars, program codes, and other software tools. In addition, physical objects such as home and office utilities, medical monitoring devices, scientific instruments and home appliances, will be represented by objects on the Internet and intranets. The duration of these objects can vary from milliseconds to years; the objects can monitor the state of processes and respond to events; and the objects can communicate with other objects. We call these objects, *active* objects. Billions of active objects will be reachable by means of the Internet. Groups of active objects will be linked together to form collaborative networks to accomplish specific tasks.

Great strides have been made in recent years on the worldwide sharing of documents (the Web), object-request brokers that support client-server applications (CORBA), enterprise-wide collaboration systems (such as those provided by Lotus Notes and Microsoft Backoffice), middleware communication packages, and most importantly, network-centric wide-area infrastructure (such as Sun's Java tools and Netscape's ONE). Investment in these areas continues to grow dramatically.

Our work exploits, but is different from, much of the work on network-centric technologies. We are concerned with the design of active object networks, layered on top of the Internet and intranets, that can support the information infrastructures of large numbers of virtual organizations and task forces concurrently. We propose to develop scalable algorithms; theories for modular design, reasoning and debugging of infrastructures for virtual organizations; and a prototype. Our work on algorithms, theories, and prototypes supports but does not directly overlap research going on elsewhere in industry and academia.

1.2 Unique Features of this Project

There is a great deal of research and development activity on Internet applications and infrastructure. In particular, much has been written about intelligent agents that roam a network searching for information (two examples are the Agent Tcl work [GCKR97] on secure mobile agents, and the TACOMA work [JvRS95] on operating system support for mobile agents). Also, much work is currently being done in industry to provide products that support global, high-speed, around-the-clock communications with asynchronous guaranteed messaging (two examples are the middleware packages Rendezvous Information Bus [TIB96] and NEONet [New96]).

This project does research that is different from research and development going on in most places in four respects:

1. **Support for Evolving Organizations and Task Forces.**

   The motivation for the computer science research that we propose to carry out is grounded on new directions in organizational structures. Specifically, organizations are becoming increasingly fluid; they adapt to situations by changing their structures through forming task forces, outsourcing work, and reorganizing rapidly. Our research deals with Internet and intranet support for rapidly evolving organizations. A specific concern is the design of a supportive infrastructure for task forces that include people and resources from several organizations.

   One of the key aspects of this project deals with workflow for evolving organizations. Workflow [Wor95] is a well-known concept that deals with how transactions flow through an organization. Ideas of workflow are related to those of intelligent mobile agents (DartFlow [CGN96] is an example of the use of transportable agents to manage workflow). However, research remains to be done, since the messages within the workflow contain information about both the data and the processing required by a transaction. The new challenge is to apply ideas of workflow to evolving organizations and emergent structures, and to demonstrate the ideas for specific task forces.

2. **Developing a Prototype.**

   We have built an early prototype using Java [GJS96], UDP [Pos80], and the World Wide Web [BLCGP92]. We propose to develop a complete prototype. We will conduct experiments with collaborators in different parts of the United States, and with collaborators outside the country. These experiments will help make preliminary evaluations about the practicality of a worldwide network of billions of persistent active objects. The thrust of these experiments will be determining the value of our ideas in supporting different types of task forces. An essential aspect of our prototype development is the use of standard, off-the-shelf software tools.

3. **Scalability.**

   Our programming model is that of networks of active communicating objects (or processes), chosen from a pool of billions of such objects distributed throughout the global information infrastructure. A key requirement for such an infrastructure is scalability. We propose to develop algorithms that scale, and to implement and evaluate our prototype.

4. **Structure.**

   A focus of our research is the structure of collaborations between active objects. Many applications have similar structures, and a research problem is to develop abstractions for these structures, develop theories for these abstractions, and then develop methods that show how the abstractions can then be reused. We are concerned with understanding collaborative structure as a formal computer science abstraction, and with practical methods for exploiting the abstraction.

   We propose to use ideas of patterns in object-oriented technology [GHJV95] to develop a library of collaborative structures, and we will show how these structures can be reused.

   We will show how to reason about collaborations between active objects, how to estimate the performance of such collaborations, and how to design (and debug) collaborative applications in a systematic way.

We propose to take steps towards achieving this vision by (i) exploiting continuing developments in commercial technologies, particularly with regard to object-request brokers and network-centric software, (ii) working with end-users of the technology to evaluate our ideas and prototypes, (iii) using Caltech's strengths in concurrent computation, and (iv) following the literature on virtual corporations and organizational development to ensure that our technology fits new directions in business structure.

Caltech's experience in distributed algorithms, systematic methods for designing concurrent systems, and concurrent object-oriented programming will be valuable in the proposed program. We propose to use our software prototypes for teaching distributed computing, and this provides added incentive to make the prototype robust and well-documented. The combination of expertise in commercial object-request broker and network technologies with expertise in distributed algorithms and systematic design is a critical factor.

1.3 Relevance of this Work

An outcome of this research are ways of exploiting the Internet and intranets for creating information infrastructure for task forces. We use a broad definition of a *task force*: it is a collection of individuals and resources that collaborate to accomplish a specific mission. Some task forces may last for hours, while others may last for years.

Military and civilian organizations are required to be increasingly flexible in responding rapidly to a variety of crises. Crises are often handled by task forces. After their missions are accomplished the task forces disband. Task forces may include units from federal, state, and city governments, private institutions such as hospitals, and even foreign institutions. Individuals may participate concurrently in several task forces. Members of a task force may be drawn from a large pool of potential members.

Trends in industry toward increasing the use of task forces or virtual organizations include greater reliance on consortia, outsourcing, and consultants. Task forces in government and academia include multi-agency crisis management operations, multi-institutional research consortia, and review panels composed of individuals from different institutions.

Setting up the information infrastructure for a task force is difficult, even if the task force itself is small, because it requires the rapid connection of the information resources of task force members selected from a wide pool. Fundamental computer science research problems in building such information infrastructures are discussed next.

2 Requirements Analysis for Task Force Infrastructure

In this section, we analyze the requirements for information infrastructures that support virtual organizations. We use the examples of virtual organizations presented in the last section to illustrate the requirements.

2.1 Infrastructures Must Adapt Rapidly

Predicting the composition of a virtual organization, far into the future, is difficult. Predicting the next crisis is difficult; so, predicting the composition of the task force that will deal with that crisis is also difficult. Likewise, identifying members of an AFOSR panel many months before the panel meets is hard because potential panel members cannot predict whether they will have conflicts of interests with proposals that will be submitted. A university consortium doing research on a project has only a short time to start work after it is funded; usually the initiators of such a consortium do not start planning for the infrastructure to support the consortium before it is funded. When a patient joins a health-care system, the patient's medical monitoring devices and information appliances must be incorporated into the information infrastructure of the system immediately. Virtual organizations incorporate new people and resources rapidly; so, their infrastructures must do so, too. Further, the structure of virtual organizations may change, and so the information infrastructure must be dynamic as well.

2.2 Infrastructures Must Be Extensible

A virtual organization grows by incoporating people and resources from a large pool distributed over some region. A crisis can occur anywhere, and so the infrastructure that supports crisis management must be able to expand anywhere. There are hundreds of people qualified to review proposals to AFOSR in university education, and they too are distributed across the country. AFOSR must be able to select any of its potential panel members, and the AFOSR information infrastructure must expand to include new members where ever they are. Likewise, universities all over the country should be able to form consortia, relying on an infrastructure that reaches all universities. Pharmaceutical delivery and laboratory tests can be contracted out to organizations that are far from patients. A virtual organization's infrastructure must be extensible to any point in the region from which members and resources are drawn.

2.3 Logical Infrastructures

The information infrastructures for virtual organizations must be logical structures, since virtual organizations are logical structures. A traditional organization, such as AFOSR, has a long lifetime and permanent resources such as employees, buildings, and computers. By contrast, an AFOSR panel has a relatively short lifetime, and relatively few resources; the panel uses the physical resources at AFOSR and members' universities. This suggests that the infrastructure for a virtual organization should be a temporary logical structure layered on top of a permanent physical structure.

2.4 Layering on the Internet

The physical structure on which the logical structures of virtual organizations are layered must reach all the people and resources that may become part of a virtual organization. We have such a physical structure: the Internet. Therefore, our problem is layering infrastructures for virtual organizations on top of the Internet. For example, the infrastructure that supports a research consortium is a logical structure that uses the Internet to span the instruments, databases, computational models, compute servers, and communication lines belonging to each of the universities in the consortium.

2.5 The Application Model

The information infrastructure for a virtual organization is a distributed system that connects active objects that serve as proxies for resources and people in the organization. From the point of view of the application developer, the active objects and connections exist until they are modified by the application. In practice, faults in the underlying system and competition for resources from different virtual organizations may require some objects and connections to be transient physically.

The idea of mapping logical constructs to hardware is not new. In the past, you had daemon processes that did various tasks for you such as check your mail. These logical processes were mapped onto a physical processor. Many processes time-shared the same processor. You could, however, treat each of your processes as though they had sole possession of the processor at all times, allowing you to focus your attention on the abstraction of a process rather than on the complexities of the physical processor on which it ran. Our problem now is similar: mapping virtual organization infrastructure onto the Internet. You can treat the infrastructures for your virtual organization as though they were permanent physical structures though the physical resources on which they are layered are shared by many virtual organizations and by other activities. You can focus your attention on the abstraction of the infrastructure of the virtual organization rather than on the complexities of the Internet, intranets, and physical resources that support it. The mapping should also take care of quality of service and fault-tolerance issues.

2.6 Scaling to Billions of Active Objects

Many telephones share resources such as communication lines because telephones are quiescent, except when used for calls. Scalability to billions of active objects may require that objects hold expensive resources except when needed. In the event of congestion where several active objects compete for the same resources, an active object may have to yield the resources it holds while it is quiescent, and then reacquire the resources when it has to deal with an event.

Active objects must be tolerant of a range of message delays (from milliseconds to days) because communicating objects may be on the same computer or on different continents.

The algorithms used for naming and resource allocation must scale. The scaling is not with respect to the size of the virtual organization, but rather with respect to the size of the pool of objects from which the infrastructure for each virtual organization is drawn.

2.7 Multiple Interfaces to Active Objects

A telephone has a single interface; all you need to call a person is that person's phone number. Objects have multiple interfaces. The object that manages proposals to AFOSR has different interfaces available to AFOSR officers, proposal authors, and reviewers. Each interface has certain rights and restrictions; for example, only the appropriate AFOSR officer can assign final evaluation scores to the proposal. Associated with each interface is a set of objects that are allowed to use that interface. The infrastructure must ensure that access to an interface is used only by objects that have the right to do so, and that accesses through an interface meet the restrictions of that interface.

2.8 Workflow for Virtual Organizations

The manner in which jobs are processed in stages by different parts of an organization is called *workflow*. For instance, workflow describes how proposals are processed in a research organization, and how invoices are processed in a purchasing organization. You can define and implement workflow for an organization today. You must also be able to define and implement workflows for virtual organizations. For instance, workflow for a multi-university research consortium on the environment can specify that measurements will be taken by a set of instruments periodically, logged into databases and fed into computational models at specified sites, and summarized into a daily report. The idea of workflow is well-known. The new requirement is that of specifying workflow for virtual organizations, and mapping the workflow from the logical infrastructure of a virtual organization to the physical layer of the Internet.

3 Proposed Research

This section gives a brief description of the basic computer science problems on which we propose to do research. We propose to work on fundamental computer science problems that support the information infrastructures of hundreds of thousands of task forces. The infrastructure for a task force is a network of objects selected from a pool of billions of communicating persistent objects. We will leverage rapid commercial developments in object-request broker and network-centric software, and focus our attention on theories, algorithms, and prototypes that support such infrastructures. We call a network of linked active objects a *session*.

Scalability. A distributed system that spans the globe must scale to billions of objects. Just as the telephone network allows you to dial any one from a pool of millions of telephones, it should be possible to set up collaborative sessions between arbitrary collections of objects selected from a pool of billions of objects. The algorithms that set up the collaborative sessions must scale to deal with large numbers of objects that can be accessed on the net, even though a collaboration may be only among a relatively small number of objects. We propose to design and implement distributed algorithms that scale to billions of objects.

Resource Management. The usage model provided by the infrastructure is that of large numbers of active communicating persistent objects. Billions of persistent communicating objects require a lot of resources. A telephone is quiescent until it is used to make or receive a call. Likewise, an object should be "frozen" and stored on secondary storage until it has to deal with an interaction, at which point the object is "thawed" to its previous state and made active. An object must remain accessible as long as any other object has a reference to it, but it should not hold expensive resources such as processes and threads except when needed for execution. We propose to explore resource management methods that scale to large large numbers of persistent communicating objects.

Varying Task Durations and Message Delays. The duration of a collaboration may vary from seconds to years. The overhead of setting up and tearing down sessions must be small to support short sessions efficiently. Furthermore, the algorithms must be resilient over a range of message delays from milliseconds to hours. We propose to design and implement distributed algorithms that are tolerant of wide ranges of message delays and task durations. We will experiment with performance tuning of prototype implementations over a range of applications and systems ranging from networks that span continents (the U.S., Europe, and Asia, to start with) to those within our lab.

Interfaces. A telephone has a single interface; all you need to call a person is that person's phone number. Objects have multiple interfaces, and different protocols may be used to access different interfaces. Setting up a collaboration between objects requires finding the objects, verifying that interfaces of objects are compatible, and then setting up the connections between interfaces. We propose to design and implement algorithms that exploit commercial technologies such as the Web and CORBA for: specifying the interface definitions for classes, implementing class inheritance, finding objects, and connecting objects into collaborative sessions.

Controlled Access to Objects. Different organizations may collaborate in some situations and compete in others. For instance, some DoD operations require the coupling of the information infrastructures of DoD units with those of foreign organizations. We propose to exploit commercial development in object-request broker technologies, and encryption, and authentication to design methods that associate access rights with each interface and ensure that objects that use an interface have the right to do so.

Methods of Providing Services to Sessions. In the past distributed systems were designed for specific applications such as air-traffic control. The new problem is that the object network supports thousands of concurrent sessions, each of which may be running different applications. Further, we do not wish to restrict the applications that sessions may execute, and so we cannot predict, at design time, what applications will be executed. We propose to: (i) identify generic services that can be provided to sessions, (ii) develop scalable algorithms for providing these services to many concurrent sessions, and (iii) implement the algorithms using commercial technologies.

An example of such a service is one that provides exclusive access. All objects in a session may need exclusive access to certain other objects; for instance, all objects in an AFOSR panel session may need exclusive access to the calendar objects of each of the panel members. Our research problem is to design and implement scalable algorithms that mediate between sessions. Our larger research goal is to extend methods in the literature for mediating between individual processes to methods that mediate between sessions of processes.

Defining Collaborative Sessions Among Objects. The rigorous specification of a collaborative session among many distributed objects is difficult for several reasons. (i) A session is a logical concept rather than a physical one. A session, unlike a telephone conference call, cannot be described in terms the resources that it holds for its duration. (ii) An object can participate in more than one session at a time, and so sessions may interfere. (iii) There are many types of collaborative sessions.

A major part of our research focus is developing methods for the rigorous specification of sessions [CR97]. We propose to use object-oriented techniques to define classes of sessions, and session templates. We must define precisely what a session is so that we can design the infrastructure that supports it.

Specifying Workflow. Workflow is a well-understood concept from office automation: it defines the flow of jobs through a network of processes. For instance, workflow can describe how proposals are processed in a research organization,or how invoices are processed in a purchasing organization. The new problem is that of implementing workflow on top of a temporary collaborative sessions of objects.

An object may be part of many sessions, each with its own workflow. The workflows of different organizations share resources. Therefore, it is not possible to design and analyze the workflow of each session independent of workflows of other sessions. The research problem is to develop computational structures that allow modular specification and design of workflows for each session.

Reasoning About Collaborative Sessions. A central research issue is developing theories and methods that help modular reasoning and debugging of session-based applications. One session is not completely independent of another if both sessions have common objects or share resources. So, it is not possible (in general) to reason about one session completely independent of others. On the other hand, reasoning about the whole global information infrastructure and its billions of objects is intractable, besides being nonmodular. We propose to develop modular methods of reasoning and debugging using rely/guarantee approaches [CS95]; the design of the session-based application guarantees certain behavior by relying on other properties of the systems. Modular design, reasoning, and debugging, are critical issues in the widespread use of worldwide object networks.

Graphical Interfaces to Support Sessions. You can make a conference call by giving an operator a list of telephone numbers. When you establish a session between communicating objects, you need to link the appropriate interfaces of participating objects. Each object may have multiple interfaces, and there may be multiple links between objects. Defining the initial session topology can be quite complex. We propose to develop graphical interfaces that allow you to draw a diagram of the objects that you want to connect in a session initially.

In many cases, once a session is initiated, it continues execution without interaction with the session initiator. There are cases, however, where you may want to control a session as it evolves. For instance, you may want to disconnect a link in the network and reconnect it elsewhere, while the session is executing. We propose to do research on user interfaces and supporting software that will allow you to control sessions dynamically.

4 Context of the Proposed Work

The amount of commercial development in network-centric applications has grown dramatically in the last few years in the United States. Microsoft OLE/ActiveX and BackOffice, IBM and Lotus Notes, Netscape LiveConnect and ONE, Sun's Java [GJS96], work in intelligent information agents [CGWK94, Whi94, JvRS95, GKN+96, GCKR97], and consortia such as the Object Management Group [Obj95] and the World Wide Web Consortium [BLCGP92] are but a few examples. At the same time, there has been explosive growth in communications technology and in support — particularly dealing with security — for commercial applications. Work on office automation and workflow [Wor95] has been going on for decades and is becoming increasingly important. All indications are that dramatic growth will continue in communications, network-centric and commercial applications.

Our research does not compete with this work. We seek to use these technologies in developing a prototype infrastructure. Developments in network-centric software (such as ILU [ILU96]) allow us to focus our effort on fundamental research on distributed computational structures, and to put relatively less effort in prototype development. Our research has immediate benefits to commercial developments because we will develop a systematic framework for design of network-centric applications.

Research on concurrent object-oriented languages [BST89, CK93] is relevant to the proposed work. A significant amount of work has been done on concurrency and persistence in distributed systems [BNOW95, Tan95]. We use this research as we explore a new direction in which sessions are treated as objects.

We continue to work in distributed algorithms [CM88, Lyn96], and we follow the research in the area, because it has a direct benefit in achieving our vision. Many of the problems listed in the previous section are problems in distributed algorithm design and implementation, except that the algorithms now have to deal with competing sessions.

The literature on systematic methods for designing sequential and concurrent programs [DS90, CT92, AL93, Cha94, Lam94, SF94, CS95] deals in large part with compositional design and reasoning: proving that a composition of components meets the specification of the composed program by using the specification (but not the implementation) of the components. Compositional design is critical to our vision as well, except that for us a component can be a session of objects; therefore components interact with each other in new ways [CR97].

In summary, we plan to use commercial off-the-shelf software where possible. We plan to exploit previous work on object-oriented systems, particularly concurrent objects, distributed algorithms, and systematic methods for designing and reasoning concurrent systems. We expect to achieve our vision of networks that span billions of communicating objects by using systems and results developed earlier. Our plan is based on continuing advances in commercial software for object-request brokers and network-centric software.

5 Tasks and Deliverables

5.1 Tasks

We describe tasks in five categories of the project: (i) theory of program structuring and reasoning, (ii) distributed algorithms, (iii) prototype implementation, (iv) distributed applications, and (v) interaction with end-users. Tasks in all categories are strongly interdependent.

5.1.1 Theory

1. **Specification of Infrastructures for Task Forces.** Develop methods for specifying sessions formally. Many task forces share the underlying inter or intranet. The same object may be part of sessions associated with different virtual organizations. Our problem is to develop methods of specifying sessions with the same degree of formality that we now specify threads.

2. **Modular Reasoning About Sessions.** Develop rely/guarantee methods that allow for modular reasoning of sessions.

3. **Sessions as Objects.** Explore using object-oriented technologies for defining classes of sessions. Investigate constructs such as inheritance for sessions.

4. **Composition of Sessions.** Develop methods for composing sessions in parallel and in sequence. One session of active objects may wait to be notified by another session, just as one thread can wait to be notified by another thread in a monitor.

5. **Specification of Workflow.** Extend methods of specifying workflow from the office-automation literature to workflow maps on top of sessions.

6. **Design Methods for Communicating Persistent Objects.** Investigate use of concepts such as indivisible tokens, distributed resource management, mutual exclusion for managing collaboration within sessions.

7. **Fault-tolerance and Quality of Service.** Develop methods for maintaining sessions in the presence of faults; specifying quality of service required by a session, and adapting the behavior of the session (changing the algorithms) when the quality of service changes.

5.1.2 Algorithms

Next we present a list of the distributed scalable algorithms that we propose to develop.

1. **Setting Up and Tearing Down Sessions.** Algorithms that link the interfaces of objects that participate in a session initially, and free up resources at termination.

2. **Providing Services to Sessions.** Algorithms that provide services to sessions such as distributed management of indivisible resources.

3. **Scalable Algorithms That Manage Large Numbers of Persistent Communicating Objects.** Performance analyses of algorithms for checkpointing and restarting objects. Design of algorithms for information discovery that scale to billions of objects.

4. **Synchronization Algorithms.** Algorithms for getting concurrent exclusive access to resources distributed across the net. Extending concepts from threads, thread synchronization, and concurrent threads within an address space to sessions of active objects, synchronization between sessions, and concurrent sessions on the Internet.

5. **Exploiting Commercial Technologies and Research Results.** These are items on which we do not propose to do basic research, but for which we propose to incorporate results and tools developed by others, including algorithms for: (i) finding and caching documents on the Web for

doing the same for communicating persistent objects, (ii) public-key encryption, and authentication to contol access to object interfaces, (iii) Java object mobility and related technologies to deal with movement of persistent communicating objects, and (iv) fault-tolerant protocols.

5.1.3 Prototype

We propose to develop a prototype using Java and sockets (TCP/IP). We will incorporate developments in Java that support checkpointing and transporting objects across the net, and remote method invocation. We will follow work on Java/CORBA interfaces and Microsoft's ActiveX technology. Though our prototype is built using Java, all the research applies to the common commercial software platforms.

The prototype will have the following features. Each person has a collection of persistent communicating objects through which the person interacts with the large number of objects on the Web. Objects are stored in secondary storage until they are needed to respond to an interaction. Each person has a list of virtual organizations (or task forces) to which the person belongs, and for each virtual organization the person's role (e.g., commander, secretary, purchasing officer) is defined. The objects and their interfaces are described in URLs. A person's objects can be found by using document search technology.

A session can be initiated by naming the objects that are to be connected initially by (i) giving their unique addresses, or (ii) giving an indirect logical address in terms of role and organization such as the calendar object of the commander of Bolling Air Force Base, or (iii) specifying attributes of the object such as "any object that lists educational Java CDs." The topology of the connections between object interfaces can be defined graphically through a GUI, or by using a template from a class of sessions. An example of the session initiation between different processes is illustrated in Figure 1.

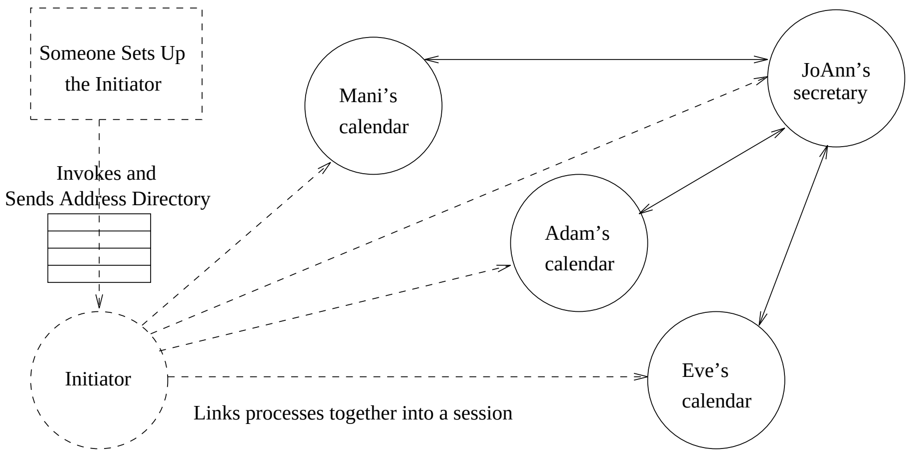

Figure 1: An initiator process uses the invoker's address directory to set up a session between existing calendar and secretary processes.

After the session is initiated, new objects can be added to the session, objects can be deleted and connections between object interfaces can be altered. Modification of the session topology is done directly by the objects themselves. Alternatively, a session can be driven by the initiator, who can disconnect and edges between interfaces and reconnect them some other way. The session structure is adaptive because the structure of the virtual organizations that it serves is also adaptive.

Each session has a collection of services that are provided to it by the infrastructure. A session can define certain objects to be indivisible tokens to be shared by participants in the session, or by participants in multiple sessions. The services provided by the session allow participants to request tokens, and guarantee that all requests are satisfied.

Graphical user interfaces are provided for defining initial session topology, workflow through participants of a session, and for defining the behavior of simple single-threaded objects.

5.1.4 Applications

We propose to develop applications in different domains to evaluate the suitability of the ideas for these domains. We will work with AFOSR in selecting problems and in the evaluation.

Crisis Management. The goal of this collection of applications is to evaluate our algorithms and prototype for their ability to rapidly create the information infrastructure and workflow needed for crisis management. One aspect of the evaluation is whether the active object network can serve as a layer on which the infrastructure for multiple concurrent crisis-management teams can be implemented. We will attempt to work with AFOSR and with organizations in Southern California who manage crisis responses. Fault-tolerance and quality of service issues for dealing with problems due to crises (nodes and links failing due to earthquakes, for instance) will be explored.

Office Automation: Proposal Review. This application defines the procedures by which proposals are evaluated by government panels as workflow over sessions of persistent communicating objects.

Many proposals at AFOSR, DARPA, and other organizations are evaluated by panels. For instance, short preproposals are evaluated by small teams of 3 or 4 people, and then by a larger panel of 20, who then recommend for or against submitting the final proposal. Final proposals are reviewed in the same manner by a different panel.

Each proposal is a task. Each panel is a session. The small teams of 3 or 4 people who initially review each proposal also constitute a session. The total panel is an example of session composition.

A program manager will be able to instruct the manager's object to set up the panels and the teams by giving the manager object the addresses of the objects of panel members. The manager will also define the workflow of transactions possibly using the graphical interface that we provide. The panels will be set up, and workflow will be monitored by our infrastructure.

Research Consortium: A Distributed Problem-Solving Environment. This application evaluates our active-object network as a layer on which the infrastructure for a collaborative problem-solving environment can be established rapidly. A problem-solving environment that we are working on presently performs climate modeling, with a short-term focus on air-quality models for Southern California. Participants in this collaborative effort may include researchers from different departments (environmental engineering, computer science, public policy), different universities (including the University of California at Irvine and Caltech), and other institutions in Southern California such as the Air Quality Management District. The kinds of workflow include flow of data (wind, temperature, pollutant concentrations) from monitoring sites to predictive models running on supercomputers to models of business, economics, and traffic in Southern California.

The experiment will help us understand the problems in laying an information infrastructure for collaboration on top of existing and planned advances on the Internet. We expect to be able to exploit high-performance networks linking many research institutions in Southern California. This research is being done with Prof. Donald Dabdub of the University of California at Irvine.

Health Delivery Systems. This experiment evaluates a virtual organization tailored to each individual consumer of health services. Consider a virtual organization that delivers health services to a student; let's call the student Luke. Suppose Luke gets a problem with what appears to be carpel-tunnel syndrome (probably from working too long on Caltech homework). Luke's personal and university insurance allows him to select doctors, radiologists, physical therapists, and pharmacies from a large pool. Before Luke got the problem, he didn't have a personal virtual organization to deliver health services to him because he didn't need one: rather, as soon as he gets the problem, he would like his own personalized organization to be set up. This requires that all the care providers (doctors, physical therapists, and so on) that he selects have an information infrastructure that links them; furthermore, workflow messages pertaining to Luke must be set up on this infrastructure. Here too, we have to set up a virtual organization very rapidly on top of our existing base which, in turn, is layered on top of the Internet.

We propose to carry out a simulation of this experiment with students playing the role of care providers because getting such an ambitious information system set up would require a change in business practices of the medical profession. The experiment would, however, give us an idea of what can be possible, assuming that all of us had interconnected information appliances.

The duration of relationships between groups of patients, doctors, psychologists, pharmacies, medical laboratories and insurance companies, will decrease as health-care services become more generic and competitive pressures increase. The information infrastructure of health-delivery systems must span medical monitoring devices and other appliances such as computers in patients' homes, cars and offices. The infrastructure must also link the providers of different medical services as they enter the system, and delete the links when they leave.

Manufacturing and Outsourcing. Flexible manufacturing systems require the manufacturing resources of different organizations to be coordinated with workflow across organizations. Tighter control of inventory, and just-in-time delivery practices, requires workflow across organizations to be controlled with precision. Often workflow is coordinated between organizations in different countries. In some industries, a company can change subcontractors frequently. Each organization may have an enterprise solution to workflow within it, but our experiment will help evaluate whether our active object network can be used to create virtual organizations that span traditional enterprises. The bandwidth required to establish workflow between clothes manufacturing companies in Bangladesh and Sri Lanka, with designers and buyers in the U.S., is not large. This experiment will help us determine how easy it is to define and set up the workflow for the virtual organizational structure.

Metacomputing. The application is to reserve resources at geographically distributed computers for a specific interval. You can specify the resources by attribute (e.g., x minutes on any IBM SP with a gigabit network connection) or by address (the Airforce IBM SP in Maui). You can specify the times at which you would like to have the resources or you can ask for the earliest possible time. The resource manager objects form a session. They communicate among each other, make the reservations if possible, and report back to you. The focus of this experiment is on evaluating scalability of algorithms for coordination.

Distance Learning, and Collaborative Learning. We propose to use our prototype for teaching a distributed computation laboratory course, CS 141, at Caltech in collaboration with other universities; our plan is to experiment with the same course material taught concurrently at the University of Florida in Gainesville by Prof. Beverly Sanders. Florida is a valuable collaborator because it is physically far (3 time zones away from Caltech). The purpose of this experiment is to use our active-object network as a basis for collaborative homeworks on distributed computing carried out by students who are indeed widely distributed.

The course is open to juniors, seniors, and first-year graduate students. The goal of the course is to have each student, in groups of 3 (2 students and a TA) publish a research technical report on a distributed application by the end of the third term, and submit many of these reports to conferences. The focus of the course is exploring the use of sessions of communicating persistent objects.

Games. Since the project has several students, including many undergraduates, it is not surprising that one of the application areas of interest is games. The application that we are working on is the implementation of asynchronous games. In a syncrhonous game, only one player can make a move at a time, and the player that can make the next move is uniquely specified by the state of the game. Bridge, chess, and Othello are examples of synchronous games. Asynchronous games have race conditions such as two players each attempting to take the card at the top of a deck. Financial market games, MUDs, and war games are examples of asynchronous games.

Implementing sessions of synchronous games is relatively straightforward. Our first application is the implementation of an asynchronous card game called "Spit" by some. The interesting features of the implementation are that you tell your object, which plays the game for you, which other objects you are willing to play with (authentication). You program the game strategy for your object. Your object can play more than one game simultaneously.

The game itself is not the issue; our concern is understanding asynchronous interaction between objects.

5.1.5 Collaboration With, and Evaluation From, End Users

Evaluation from end-users is a critial part of our research agenda.

We expect to work with government research agencies, such as AFOSR, Rome Laboratories, and DARPA, that evaluate proposals. We hope to have these agencies evaluate workflow over sessions of objects. We have a collaboration with a consortium of universities (Indiana, Drexel, New Mexico State) and Los Alamos national laboratories on problem-solving environments. We expect to work with this consortium in evaluating ideas for connecting physical objects (medical monitoring devices, scientific instruments) within a problem-solving environment.

We have plans to work with Hewlett-Packard research laboratories on evaluating algorithms for synchronization, with the specific application being scheduling and managing of videoconferences. Collaborations with professors at some universities, such as Prof. Beverly Sanders at the University of Florida, will help us evaluate our prototype software as a a teaching tool. The AFOSR Center for Research in Parallel Computing and its researchers in applications will help to evaluate the algorithms for resource reservation in metacomputing.

5.2 Deliverables

Year 1. 1. Theory for formal specification of the information infrastructures for task forces. We call these infrastructures *sessions* in keeping with the telephone analogy.

Abstractions for sessions as objects. Abstractions for patterns of sessions.

2. Abstractions for synchronizing between sessions. We will extend the abstractions and theory of concurrent threads sharing an address space to concurrent sessions sharing an internet or intranet. Past research has developed constructs for thread creation, waiting, notify, suspension, and resumption; this deliverable will be a set of constructs for session creation and termination, one session waiting for other sessions, session supension and resumption, and so on.
3. Version of prototype that includes initiator control of sessions and use of Web technology for finding objects and declaring object interfaces. Graphical user interfaces for defining initial session topology.
4. Release of a prototype collaborative problem-solving environment and games applications.
5. Evaluation by problem-solving environment consortium dealing with a specific application such as air quality monitoring and modeling in Southern California.

Year 2. 1. Modular reasoning about sessions, and session composition theory.

2. Algorithms for providing services to sessions: distributed management of indivisible resources (tokens), distributed clocks, synchronization within sessions.

   Algorithms that exploit commercial technologies for: (i) finding and caching documents on the Web for doing the same with communicating persistent objects, and (ii) security for controlling access to interfaces of objects.

   Database structures to keep track of virtual organizations and roles in virtual organizations, and allow objects and interfaces to be specified in terms of organizations and roles.
3. Prototype that incorporates these services to sessions. Graphical user interfaces for defining simple active objects. Prototype includes workflow layered on top of sessions. Graphical interface to allow initiator of a session to control a session dynamically.
4. Release of the office automation application and a simulation of the flexible manufacturing application.
5. Evaluation by a government agency such as AFOSR of workflow layered on sessions.
6. Document describing approaches to provide quality of service and fault information from the underlying infrastructure to sessions. Examples of specific methods for sessions to adapt to information provided by the underlying infrastructure. Evaluate the approaches in the context of a crisis management application with earthquakes and fires damaging underlying intra and internets.
7. Document describing initial approaches to scaling the information discovery and caching problems when the number of objects scales far beyond billions of objects. If each person has a personalized virtual organization for health delivery to that person, then the number of virtual organizations supported by the internet is in the hundreds of millions worldwide, and the number of objects can approach trillions. This document evaluates current strategies in the light of scaling of both the number of concurrent sessions and the number of objects.

Year 3. 1. Theories for modular specification and reasoning about workflows. Book on design methods for communicating persistent objects.

2. Algorithms that exploit commercial software for object mobility and fault tolerance for use in sessions of communicating objects.

3. Robust implementation that allows: object migration, specification of objects by attribute and composition of sessions. Graphical user interfaces for defining workflow.
4. Release of the metacomputing and health-delivery applications.
5. Implementation that allows a distributed application to adapt to quality of service and faults in the underlying system.
6. Evaluation by universities of the infrastructure as a means for teaching distributed computing in a distance-learning and collaboration environment.

   Evaluation by companies of the prototype implemented on their platforms for office automation applications.

6 Research Completed on Active Object Networks

This section describes the prototype implementation work carried out so far and design directions for the future and later discusses theoretical and algorithmic issues. This section has three subsections: (i) a description of the prototype design and version 1.0 implementation, (ii) algorithmic issues, and (iii) theoretical issues dealing with reasoning about the correctness of networks of active objects.

6.1 Networks of Active Objects

Next, we present the requirements for networks of active objects. These requirements influence our design. We will show how each aspect of our design is influenced by these requirements.

6.1.1 How Requirements Influence the Design

Next, we describe how the requirements influenced our design. The design is given later, in detail. Here we provide only the motivation for our design choices.

Motivation for Messages. Active objects carry out tasks for individuals or groups. They are proxies for people and resources. Workflow in virtual organizations is often defined in terms of messages (tasks) that flow from person to person for processing at each stage. Therefore, in our design, workflow is defined in terms of messages flowing from active object to active object.

Motivation for Typed Authenticated Message Boxes. Virtual organizations have structure — for example, the accounting department prepares a daily report to the president — and we want to create information infrastructures that reflect the structures of their virtual organizations. Therefore, the information infrastructure must be more than a standard distributed system consisting of processes and communication channels; the structure must support different kinds of interfaces and different kinds of connections. Interfaces should have types and a set of active objects authorized to communicate through that interface, just as only certain people are authorized to call the president on the emergency phone line.

Motivation for Asynchronous Communication. The message-passing layer has acknowledgments for messages; however, the programming model is asynchronous message passing: an active object can repeatedly send messages without waiting for acknowledgments for earlier messages. This is because message delays to different points on the global information infrastructure, and responses from appliances or humans can vary widely; a communication model based on synchronous RPC's can result in active objects remaining blocked for arbitrary periods of time. The programmer can write programs that wait for acknowledgments or that do not wait but expect exceptions to be raised if messages are not delivered.

Motivation for Peer-to-Peer Communication. In a virtual organization, people and resources can communicate with each other directly without necessarily going through a "broker." Also, active objects and channels can persist indefinitely, and they participate in an ongoing long-term communication. The physical model is that of peers and channels of communication that persist; therefore, a peer-to-peer communication model reflects the structure of a virtual organization better than client-server and object-request broker (ORB) models. The underlying layer may be implemented using client servers or ORBs, but the programming model is peer-to-peer.

Overview of How Requirements Influence Design. Active objects carry out tasks for individuals or groups. They are proxies for the people and resources they serve. Workflow in virtual organizations is often defined in terms of messages (tasks) that flow from person to person for processing at each stage. Therefore, in our design, workflow is defined in terms of messages flowing from active object to active object. As illustrated in Figure 2, our active objects have inboxes and outboxes for messages just as people do. The network layer ensures that messages are delivered to an inbox only from objects authorized to do so.

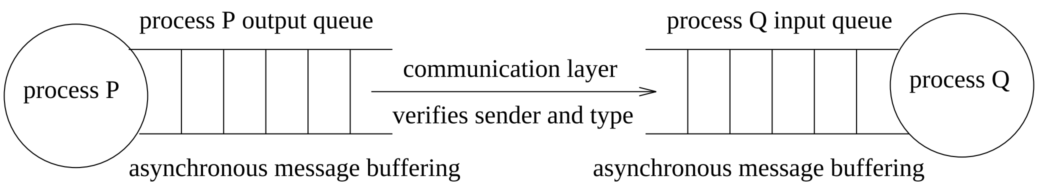

Figure 2: To modify the state of remote process Q, process P sends requests asynchronously.

Addesses and interface definitions of inboxes can be found on the Web. Active objects are implemented as processes (in Unix or NT) terms, but an active object yields its process slot when it does not need to execute; thus even a personal computer may have hundreds of active objects that timeshare process slots.

We will show later that the software infrastructure for active object networks is both simple and small, and it has the functionality to support mobile intelligent agents.

Next, we discuss the design in greater detail.

6.1.2 Communication

Interaction by Message Passing. An active object interacts with other active objects by sending and receiving messages. Different active objects do not share state. A message can be a method (in object-oriented terminology) and can serve as a one-way or asynchronous RPC or remote-method invocation.

Mailboxes. Associated with each active object is a set of objects called mailboxes. Each mailbox is either an inbox or an outbox. An active object sends messages via its outboxes and receives messages from its inboxes. An active object can connect any of its outboxes to a set of inboxes of its own or other active objects; the connection creates a first-in-first-out error-free directed communication channel between the outbox and each inbox to which it is connected. Message delay in a channel is arbitrary; if a message is not delivered within a specified time to its destination inbox an exception is raised in the sender.

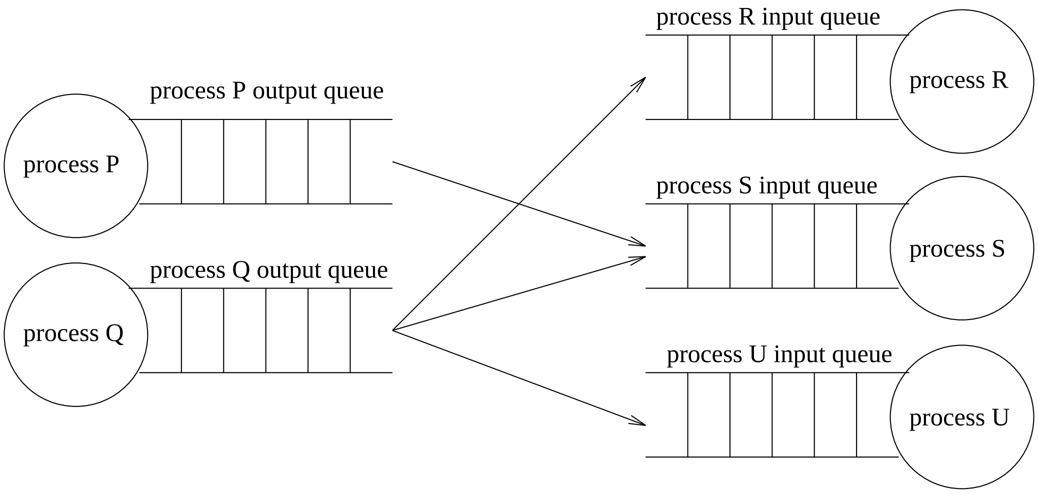

Figure 3: An example of process input and output queue connections: process P's output queue is bound to process S's input queue; process Q's output queue is bound to broadcast to the input queues of processes R, S, and U. Messages are fairly merged on process S's input queue. Our message-passing mechanism ensures FIFO delivery of messages on any given channel.

As illustrated by Figure 3, since an outbox can be connected to an arbitrary number of inboxes, there may be arbitrarily many channels from an outbox. When a process sends a message on an outbox, a copy of the message is sent along each channel from that outbox. There can also be many channels feeding into an inbox; the sequence of messages delivered to the inbox is a fair merge of the sequences of messages sent to it along each of its incoming channels. Each channel is uniquely specified by the id's of the sending object, its outbox, the receiving object and its inbox.

Connections between Outboxes and Inboxes. Each mailbox has a type which specifies the types of messages that can be sent or received on that mailbox. When an active object connects one of its outboxes to a set of inboxes the connection takes place only if the types of outbox and inboxes are compatible: every message sent along the outbox of a channel must be acceptable at the inbox at the end of the channel. If an active object's attempt at establishing a connection fails, an exception is raised within the object. As shown by Figure 4, for efficiency reasons, the implementation has two kinds of connect operations: a safe connect described above and a quick connect that does not check for compatibility of types. We do not discuss quick connect here in detail because it is not critical for the discussion.

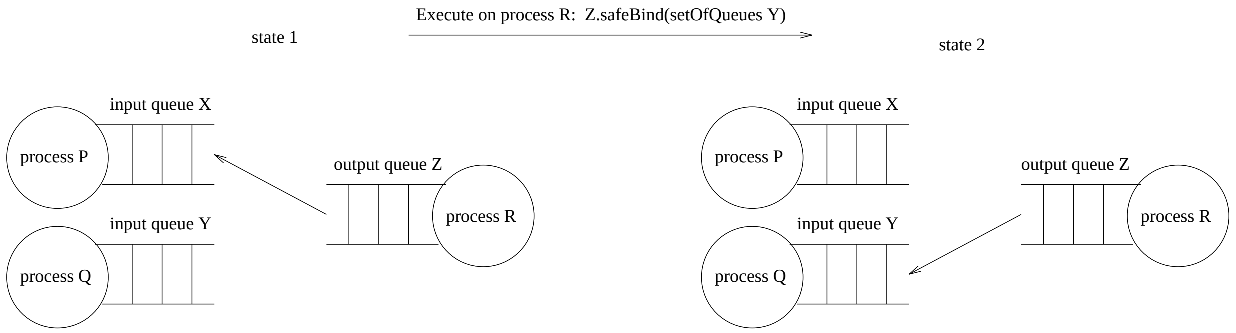

Figure 4: In changing the binding of an output queue to a different set of input queues (or, in this case, to a different single input queue), a process can perform a quick connect or a safe connect. Note that the process that has the output queue changes the binding, using an asynchronous request. The binding mechanism ensures reliable reconnections when a safe connect is called.

Authorization. Each inbox `ib` has two sets of associated object id's: `ib.accept` and `ib.reject`. An active object `x` is authorized to send messages to inbox `ib` if and only if `x` is in `ib.accept` and not in `ib.reject`. The safe connect method checks that the sender is authorized to connect to the designated inbox and raises an exception otherwise. The sets `ib.accept` and `ib.reject` are specified as an explicit list of object id's and an implicit list of attributes of object id's such as portions of the Internet address (`mil.af.afosr.*`); any object is a member of the set if and only if it is in the explicit list or it has any of the attributes of the implicit list.

Dynamic Structure of Computations. Active objects, mailboxes and connections are dynamic: they can be created, deleted and altered during program execution. The values of the sets `accept` and `reject` can also be modified dynamically.

Sending and Receiving Messages. An outbox has a method `send`; execution of `ob.send(m)` causes copies of message `m` to be sent on each of the channels outgoing from outbox `ob`. This method does not suspend. An exception is raised by a systems-management thread if a message cannot be delivered. An inbox has methods to check whether it is empty and to receive messages from the inbox.

The underlying message-passing structure is asynchronous because because message delays on the global information infrastructure can vary a great deal. Synchronous communication which requires round-trip messages can be very slow if one object is communicating with another in a remote part of the world. Synchronous communication is, however, supported by having the sender suspend on a designated inbox until an acknowledgment message is received on it.

Messages. There is a base message class with methods `toStream` and `fromStream` that convert the message to a stream and reconstruct a message from a stream. You override these methods for your message classes.

6.1.3 Persistence

Active objects classes are derived from an abstract base class that has abstract methods, `freeze` and `thaw`. A thread in the active object calls the freeze method which causes the state of the object to be serialized, i.e., converted into a stream, and stored in secondary storage. The thaw method is called by the system when a message arrives in the mailbox of a frozen object. The thaw method causes the object to be instantiated with the values of variables at the point at which the object was last frozen. In the current implementation, the states of the object at the point at which it is frozen and the point at which it is thawed are not identical because the program counter after the thaw is at the start of the `main` program, not immediately after the call to `freeze`. In future implementations we plan to explore the implementation in which the program counter after the thaw is at the point after the call to the corresponding freeze.

An active object is a process (in UNIX terms) after it is thawed and before it is next frozen. While the object is frozen, it does not occupy a process slot; however, from the point of view of the programming model there is no difference whether the object occupies a process slot or is stored in secondary storage.

6.1.4 The System

Control Threads within Active Objects. As illustrated in Figure 5, an active object has an address space and one or more threads. Different active objects do not share state. The threads within an active object share a common address space. Thus the overall model is hierarchic with message passing between active objects and a single address space shared by threads within an object.

We are doing research on thread patterns such as: objects with a single thread that polls its inboxes, objects with one thread per inbox, and objects with high-priority threads to deal with urgent messages from the system and low-priority threads to deal with normal messages.

Composing Distributed Objects. The composition of distributed objects requires checking if their interfaces match; once that checking is complete, a session connection can be set up, and a series of transactions can occur. For example, as shown in Figure 6, independently running distributed processes can collaborate on-the-fly over the Internet safely, because their interfaces have been verified to be appropriate.

Adapting Applications to the Environment. A worldwide object network is dynamic: communication links are added, the utilization on links changes, and machines fail. One of our research objectives is to design applications that adapt to the environment. At the simplest level, information about the environment is obtained by making straightforward measurements such as the time between sending a message and the underlying message layer (in the sender) receiving an acknowledgment that the message has been delivered to the appropriate inbox: the application can adapt as this delay changes.

Each active object has designated inboxes through which it receives system information. All active objects inherit from the same base class which has inboxes designated for commands from the system, as illustrated in Figure 7.

Information about the environment is given to the application by the system and the application then adapts. One of the research issues is developing algorithms for distributed applications to adapt: the problem is that each active object can get different information about the environment from its local object manager. Thus, one object can adapt one way while other objects in the same application

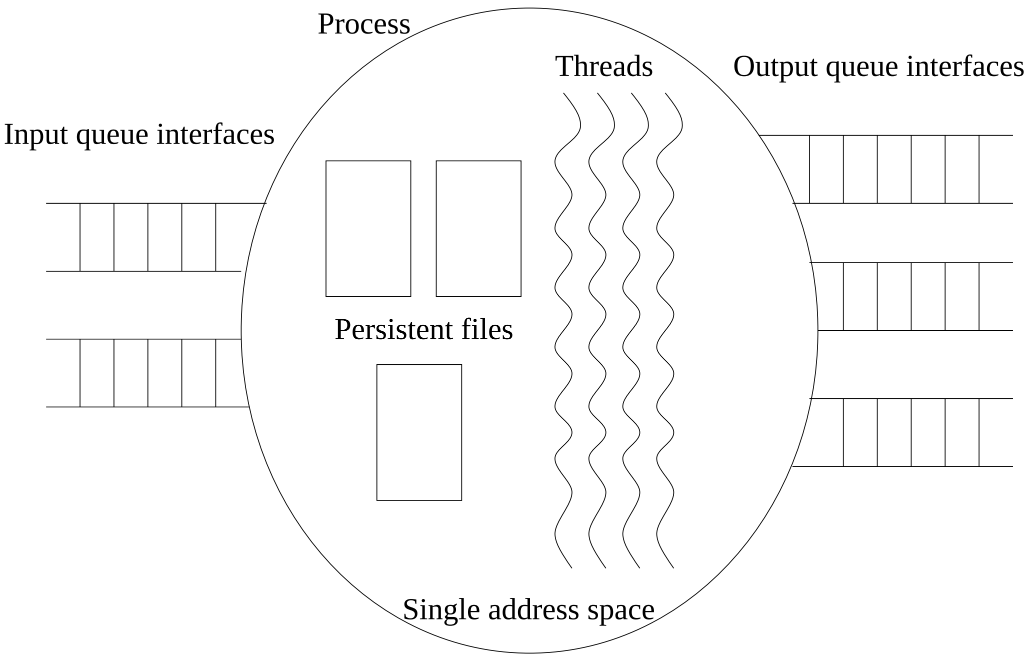

Figure 5: State is encapsulated in the persistent files of an entity's corresponding process. A process is a persistent object that may be multithreaded. Interfaces to other processes provide a mechanism for reliably transferring requests to modify local state.

adapt in a different way causing chaos. Our research objective is the development of algorithms that allows a distributed application to adapt to its changing environments in a coordinated fashion.

6.1.5 Systems Services to Active Objects

An important research problem is that of providing services to active objects even though we do not know what applications the objects are running. To give a flavor of the kinds of services required by networks of active objects we give two examples. Important research questions are: (i) defining an interface between the underlying system and the application across which the application can ask for services and the system can provide services, and (ii) determining what core portfolio of services to provide.

Mutual Exclusion Between Distributed Applications. Consider the following problem. Assume that you are a member of several task forces such as serving on an AFOSR panel and also serving on a university curriculum committee. Each of the task forces needs to arrange a time for a videoconference. One convenient way to determine such a time is for each task force to gain exclusive access to the calendar objects for *all* the members of that task force. If the AFOSR panel gains exclusive access to the calendars of all panel members then determining the eariest possible meeting time is straightforward. The difficulty is that resolving mutual exclusion between two sets of distributed applications is not merely adjudicating between two objects but two *sets* of distributed objects.

This example illustrates some of the problems of designing the service layer. It must provide mutual exclusion between sets of active objects where each active object requires exclusive access to

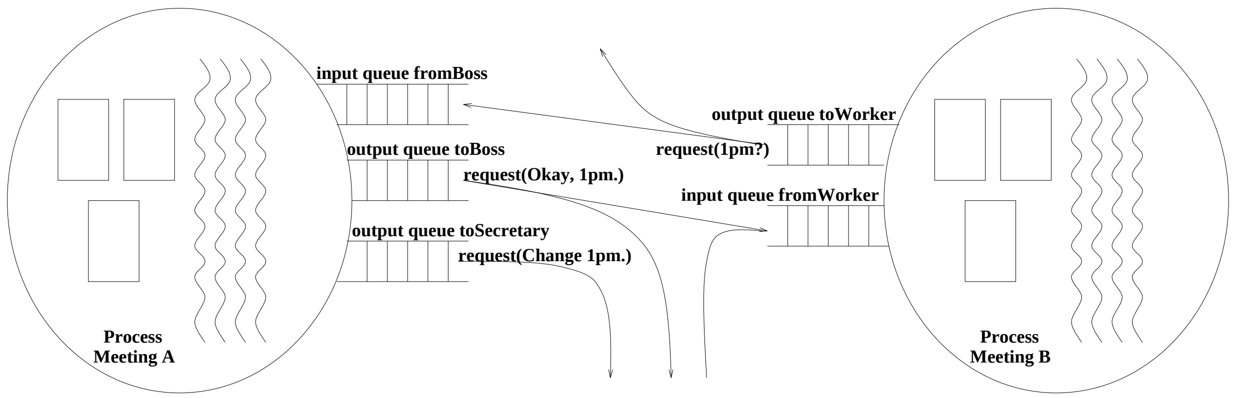

Figure 6: Sessions can be set up on-the-fly once interface matches have been checked. Here, a boss can have her meeting process request a 1pm get-together with her workers, and the workers' processes can receive that request and (automatically or interactively) reply, also requesting to their secretaries to change their 1pm appointments, if needed.

some specified set of (usually local) resources. The interface must allow the application to request the service layer to lock any arbitrary set of distributed resources. Scalable algorithms to implement this service are discussed in the section on algorithms.

Sharing Indivisible Resources within an Application. The previous example illustrated sharing of distributed resources between applications. This example deals with sharing indivisible resources by active objects within the same application.

Suppose several active object within an application for an AFOSR task force need exclusive access to a set of files. The problem is that of sharing exclusive access to the files among all the objects that need them, in a fair way. This problem is also so common that it is part of the core portfolio of services that we provide. The approach we take is to define an object class called token. A token is indivisible; it can be passed from object to object (through the service layer) but it cannot be duplicated or destroyed except by methods executed within the service layer. Each token has a "color." There can be several tokens of a given color. Tokens needed by an application are created when the application is instantiated. The interface between the application and service layer allows an object to request sets of tokens, e.g., 5 red tokens and 3 green tokens. The service layer guarantees to get the requested resources to the requestor provided all objects release the resources that they get in finite time, and do not request additional resources until they release the resources they have.

These two little examples, and the paragraphs on how applications can adapt to the environment, illustrate some of the research issues in designing the interface between the application and service layers. The interface must be generic because we do not know what applications will be run. However, the interface must be capable of being tailored to serve a specific application.

6.1.6 Exploiting the Web

As shown in Figure 8, you can exploit Web search engines to find addresses of (inboxes of) objects to which you can connect outboxes of your objects. Active objects have URLs, just as passive objects do. An active object is designed to have some permanent inboxes and some temporary inboxes that

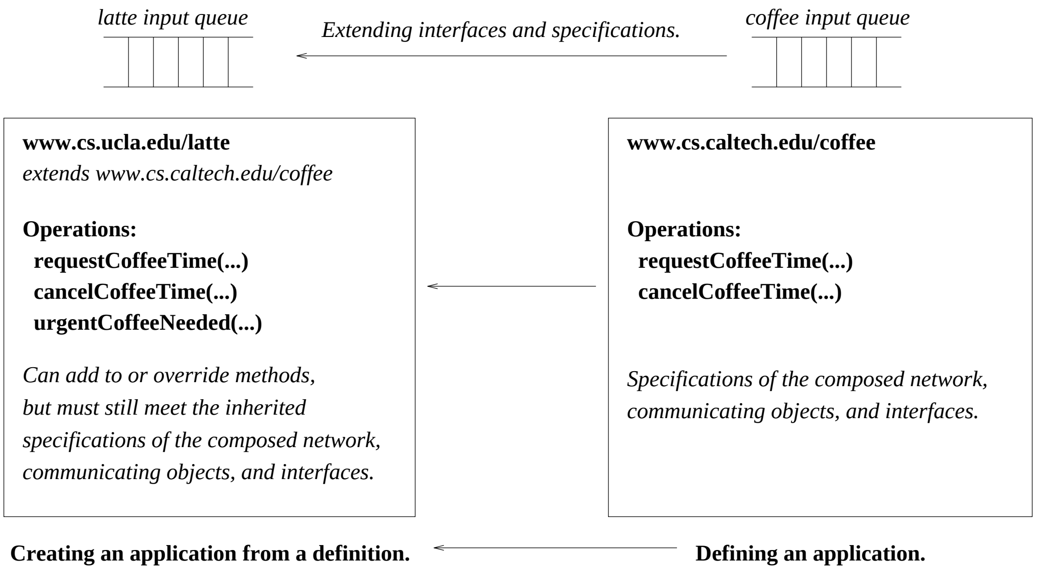

Figure 7: Inheritance types are compatible only if the interfaces are inherited from the same URL (Universal Resource Locator). In this sense, we exploit existing Web technology.

are created and deleted as need arises. The addresses of the permanent inboxes are available in the object's URL. For instance, you may know that the chair of your AFOSR task force has an inbox called `AFOSRPanel` to his or her calendar object. You can find the address of that inbox by starting at the chair's home page. The methods that are accepted by that inbox (i.e., its interface definition) is also described in the document for that URL. You can then connect an outbox of your calendar object to that inbox, ensuring that the types of the out and in boxes match.

The Web and its search engines provide immense power in finding objects and getting the necessary documentation to connect to them.

6.2 Algorithms

The next section describes some of the research on scalable algorithms that support worldwide networks of active objects. New algorithms are required to deal with scaling to billions of objects, vast variation in message delays, and objects that do not behave as they should. To give a flavor of the algorithmic issues we discuss one of the example services discussed earlier.

Algorithms for Mutual Exclusion between Applications. Next, we present a traditional solution to a mutual exclusion problem and then discuss problems with scaling the algorithm for the global information infrastructure. The example given earlier of two task forces, an AFOSR panel and a university committee, is helpful. From time to time each object servicing a task force requires exclusive access to local resources, for instance, the AFOSR panel requires exclusive access to the calendar objects of each of its members. One algorithm is that locks on the objects are acquired in a specified sequence such objects ordered by id. Since object id's are strings, a universal total ordering of objects is obtained merely by ordering their id's. So, the AFOSR panel gets a lock on the calendar object of one member, and then the next in the sequence and then the next, and so on. If a desired

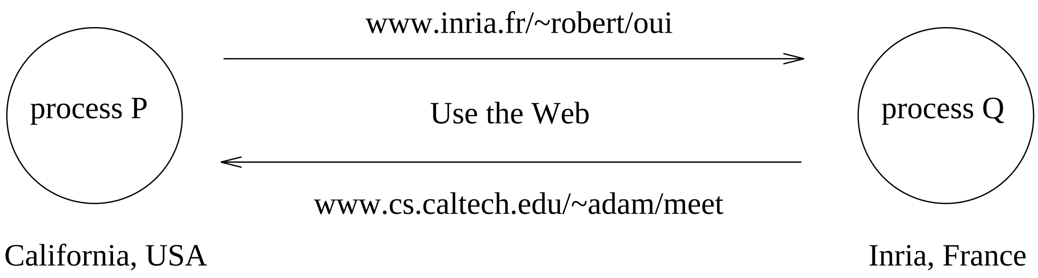

Figure 8: The Web can be used to locate the interfaces and specifications of distributed (and possibly mobile) processes. The Web can also be used to locate the running programs themselves.

calendar object is locked by some other group, the AFOSR panel application waits. The total ordering on acquisition of resources guarantees absence of deadlock (provided once each application gets all the resources it needs, it releases them eventually).

What are the difficulties with scaling this algorithm? The scaling problem is *not* that a task force becomes increasingly large, but that the task force's infrastructure is layered on a global infrastructure that is shared with many other organizations, and it is the global infrastructure that is becoming increasingly large. Suppose the computation for the AFOSR panel to acquire exclusive access to calendar objects of its members is waiting to gain access to Major Luginbuhl's calendar object which is now locked by a university curriculum panel. The computation to acquire locks on calendar objects of all members of the university curriculum panel may be suspended waiting to gain access to Prof. Fred Schneider's calendar object which is now locked by a program committee for a distributed agents workshop which is waiting to get a lock on Prof. Tony Hoare's calendar object in Oxford. It is easy to see that the chain of waiting can, quite easily, become large and stretch around the world. Even though the algorithm is deadlock free, it is susceptible to very long waits and congestion anywhere along the waiting chain.

An alternate solution is to have a single server responsible for locking all resources worldwide. This solution has the obvious difficulty that the server is a bottleneck, and the server is remote from objects in other continents. This approach has the advantage that global decisions can be made and global decisions make more effective use of resources than series of local decisions.

One of the strategies that we are exploring is to use a hierarchical approach. Consider a tree based on URLs spanning all objects. Examples of nodes in the tree are `cs.cornell.edu` and `cs.caltech.edu`. As illustrated in Figure 9, resources are acquired in order of nodes of the tree: for instance, lock all resources required from `cs.caltech.edu` before locking any resources from `cs.cornell.edu`. Each node makes decisions based on information about its domain. This approach does not avoid the problem of waiting chains stretching round the world but it does make the chain structure more efficient (because you won't have a chain from Caltech to Bangalore to Cornell to Dublin to Dartmouth) and shorter. We expect to carry out experiments evaluating the performance of different strategies and to do basic research on algorithm development.

One of the algorithms that we are evaluating for scalability uses time stamps and tokens. Tokens cannot be destroyed or duplicated. Each resource with exclusive access is represented by a token. In the example of locking calendars, each calendar object is represented by a token. The head (active object) for a virtual organization, such as the university committee or AFOSR panel, can request all the tokens corresponding to calendar objects of its members. (Note that calendars do not actually

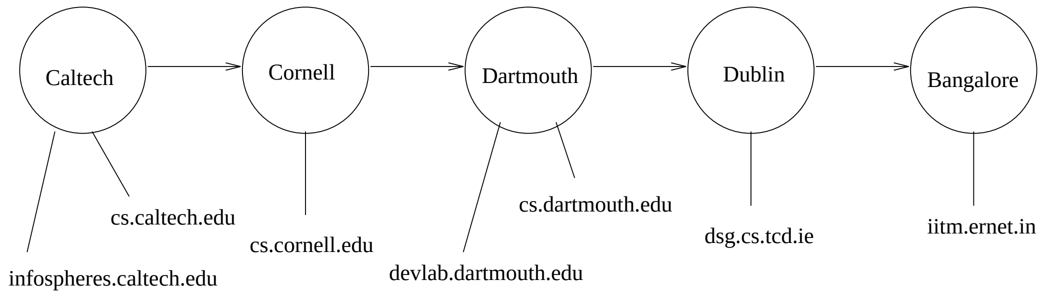

Figure 9: One strategy is to acquire resources in a specified order, both to avoid deadlock and to garner efficiency.

move; only their representative tokens do.) Each request for a token is time-stamped with the time that the request is made. The essence of the algorithm is that tokens flow to the request with the earliest time stamp. Ties between timestamps are broken by using the id of the sender. The research we propose to do will predict the performance of these algorithms and we will run experiments on a global scale to evaluate the performance predictions.

We have discussed this problem at some length merely to give a flavor of the kinds of algorithmic research issues that we propose to investigate; the problem itself is not central to our project.

6.3 Theory

6.3.1 Reasoning

The active objects and connections for each task force can be viewed as a distributed system. Thus, the underlying infrastructure supports many distributed systems concurrently because it supports many task forces concurrently. We would like to design and reason about each such system by itself. The systems interact because they may share common resources. For instance, the distributed systems for the AFOSR panel and the CS Visiting Committee share calendar objects of its members. Since these systems are not completely independent we cannot design them ignoring all the other systems. We cannot, however, design the entire global information infrastructure with its billions of active objects as a single monolithic entity. The research issue deals with designing and reasoning about a distributed system for a task force in a modular fashion: each task force relies on some properties provided by the underlying infrastructure and in return, each system guarantees certain kinds of properties that the infrastructure can rely upon. Our research is focused on developing new rely/guarantee methods of design and analysis that help in modular design of collaborations.

6.3.2 Abstractions of Collaborative Structures

The analogy between the telephone network and the network of active objects relates a telephone conference call to a collaboration between active objects. A conference call has very simple structure. By contrast, collaborations between active objects may have very complex structures, which are represented (in part) by the objects and connections between inboxes and outboxes. Furthermore, the evolution of a conference call is straightforward: people can come in or drop out of the session. By contrast, the evolution of a collaborative structure can be extremely complex. Many applications have similar collaborative structures, just as many organizations have similar structures. Our research deals with discovering abstractions for these structures that can be reused for different applications. The idea is to use concepts from patterns of objects to deal with patterns of collaborative structures, to develop methods for analyzing the correctness and performance of archetypal structures, and to build libraries of structures that can be reused. Similarly, we will develop abstractions to capture patterns of workflow and demonstrate systematic design of workflow on top of emerging systems by reusing the abstractions.

7 Previous Work by the Team

The Caltech team has a track record in concurrent computing. We have done pioneering work on systematic design of concurrent systems with our participation in the development of UNITY [CM88]. Our work on compositional concurrent programming languages [CT92, Cha94] has led to ideas and implementations (e.g., CC++ [CK93] and Fortran M [FC92]) that are widely used. The team participated in some of the earliest work on performance modeling [CS80] and parallel simulation [CM79]. The Caltech work on parallel archetypes [CMMM95] is being investigated in several projects. We did some of the early work on structures for creating and synchronizing among threads in Java [SC96].

We have begun an initial implementation of the infrastructure to support a worldwide network of persistent communicating objects [CR97]. Our paper describing this effort won the best paper award at the 1996 High Performance Distributed Computing Conference [CRS+96]. The PI won the A.A. Michelson Award for contributions to performance modeling and distributed simulation, the IEEE Koji Kobayashi Award for contributions to computing and communication, and is a member of the National Academy of Engineering.

References

[AL93] M. Abadi and L. Lamport. Composing Specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993. Also available as DEC SRC Research Report 66.

[BLCGP92] T. Berners-Lee, R. Cailliau, J. Groff, and B. Pollermann. World Wide Web: The Information Universe. *Electronic Networking: Research, Applications, and Policy*, 1(2), 1992.

[BNOW95] A. Birrell, G. Nelson, S. Owicki, and E.P. Wobber. Network Objects. *Software Practice and Experience*, 25(S4):87–130, December 1995. Also available as DEC SRC Research Report 115.

[BST89] H.E. Bal, J.G. Steiner, and A.S. Tanenbaum. Programming Languages for Distributed Computing Systems. *ACM Computing Surveys*, 21(3):261–322, September 1989.

[CGN96] T. Cai, P. Gloor, and S. Nog. DartFlow: A Workflow Management System on the Web Using Transportable Agents. In *Submitted to PDIS 1996*, Dartmouth College, 1996.

[CGWK94] G. Cybenko, R. Gray, Y. Wu, and A. Khrabrov. *Information Architecture and Agents.* Technical Report Work in Progress, Dartmouth College, 1994. Copyright June 1994.

[Cha94] K.M. Chandy. Properties of Concurrent Programs. *Formal Aspects of Computing*, 6(6):607–619, 1994.

[CK93] K.M. Chandy and C. Kesselman. CC++: A Declarative Concurrent Object-Oriented Programming Notation. In *Research Directions in Concurrent Object-Oriented Programming*. MIT Press, 1993.

[CM79] K.M. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979.

[CM88] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, Reading, MA, 1988.

[CMMM95] K.M. Chandy, R. Manohar, B.L. Massingill, and D.I. Meiron. Integrating Task and Data Parallelism with the Collective Communication Archetype. In *Proceedings of the International Parallel Processing Symposium IPPS-9*, April 1995.

[CR97] K.M. Chandy and A. Rifkin. Systematic Composition of Objects in Distributed Internet Applications. In *Proceedings of the 30th Hawaii International Conference on System Sciences*, Hawaii, January 1997.

[CRS+96] K.M. Chandy, A. Rifkin, P.A.G. Sivilotti, J. Mandelson, M. Richardson, W. Tanaka, and L. Weisman. A World-Wide Distributed System Using Java and the Internet. In *Proceedings of the Fifth Workshop on High Performance Distributed Computing*, Syracuse, NY, August 1996.

[CS80] K.M. Chandy and C. Sauer. *Computing Systems Performance Modeling: A Primer*. Prentice-Hall, 1980.

[CS95] K.M. Chandy and B.A. Sanders. Predicate Transformers for Reasoning about Concurrent Computation. *Science of Computer Programming*, 24(2):129–147, April 1995.

[CT92] K.M. Chandy and S. Taylor. *An Introduction to Parallel Programming*. Jones and Bartlett, Boston, MA, 1992.

[DS90] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, New York, 1990.

[FC92] I. Foster and K.M. Chandy. *Fortran M: A Language for Modular Parallel Programming*. Technical Report CRPC-TR92249, Center for Research on Parallel Computation, 1992. Revised June 1993.

[GCKR97] R. Gray, G. Cybenko, D. Kotz, and D. Rus. Agent Tcl. In *Itinerant Agents: Explanations and Examples with CD-ROM*, W. Cockayne and M. Zypa, editors, Manning Publishing, 1997.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley Developers Press, Sunsoft Java Series, 1996.

[GKN+96] R. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko. *Mobile Agents for Mobile Computing*. Technical Report PCS-TR96-285, Dartmouth College, 1996. Submitted to MobiCom 96.

[ILU96] ILU Group. *Inter-Language Unification.* Xerox Parc, Palo Alto, CA, 1996.

[JvRS95] D. Johansen, R. van Renesse, and F.B. Schneider. Operating System Support for Mobile Agents. *Proceedings of the Fifth IEEE Workshop on Hot Topics in Operating Systems*, HOTOS-V:42–45, May 1995. Orcas Island, Washington.

[Lam94] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994. Also available as DEC SRC Research Report 79.

[Lyn96] N.A. Lynch. *Distributed Algorithms.* Morgan-Kaufmann, San Francisco, CA, 1996.

[New96] New Era Of Networks, Inc. *NEONet.* Technical report, Available on the Web at `http://www.neonsoft.com/neonet.htm`, 1996.

[Obj95] Object Management Group. *The Common Object Request Broker: Architecture and Specification (CORBA).* OMG, 1995. Revision 2.0.

[Pos80] J.B. Postel. *User Datagram Protocol.* RFC 768, August 1980.

[SC96] P.A.G. Sivilotti and K.M. Chandy. Toward High Confidence Distributed Programming with Java: Reliable Thread Libraries. In *International Conference on Software Engineering*, Las Vegas, NV, July 1996.

[SF94] F.B. Schneider and L. Fix. Reasoning About Programs by Exploiting the Environment. *Proceedings of the 21st International Colloquium, ICALP 94*, pages 328–339, July 1994. Jerusalem, Israel; Lecture Notes in Computer Science 820, Springer-Verlag, New York.

[Tan95] A.S. Tanenbaum. *Distributed Operating Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[TIB96] TIBCO, Inc. *Rendezvous Information Bus.* Technical report, Available on the Web at `http://www.rv.tibco.com/`, 1996.

[Whi94] J.E. White. *Telescript Technology: The Foundation for the Electronic Marketplace.* General Magic, Inc, 1994.

[Wor95] Workflow Management Coalition, International Organization for the Development and Promotion of Workflow Standards. *Workflow Glossary.* Workflow Management Coalition, Avenue Marcel Thiry 204, 1200 Brussels, Belgium, 1995.