Network Computing

Munindar P. Singh* and Mladen A. Vouk†
Department of Computer Science
North Carolina State University
446 EGRC/Box 7534, 1010 Main Campus Drive
Raleigh, NC 27695-7534, USA

singh@ncsu.edu, vouk@csc.ncsu.edu

*Munindar Singh's research is partially supported by the NCSU College of Engineering, the National Science Foundation under grants IRI-9529179 and IRI-9624425 (Career Award), and IBM corporation.

†Mladen Vouk's research is partially supported by the National Science Foundation under grant ACS-9696131, NASA, EPA, and IBM corporation.

Contents

1 Introduction

1.1 What is Network Computing?

The rapid expansion of the computing and communications infrastructure is not only enabling new grand applications, but also influencing the science of computing in terms of

- the metaphors with which we view computing
- the architectures of computing systems
- the specific technologies that realize those architectures
- the tools and techniques with which we construct them.

The term often used for this new paradigm is *network computing* (NC). NC environments (NCEs) are envisioned as collections of interactive and cooperating programs, tools, clients, and intelligent agents. These components are (ultimately) integrated into an "appliance-like" environment that facilitates users' (professional, learning, entertainment) activities by maximizing active, friendly support for them and minimizing their interactions with the underlying systems.

For example, consider a Network-Based Education (NBE) system, a typical NC application. The prime function of an NBE system is to maximize knowledge transfer and retention to a large number of students distributed in both time and space. The student user is primarily concerned with the availability and quality of the educational material, often at times other than the ones when the lectures are given. The instructor, another type of NBE user, deals with both in-class students and "live" remote students who attend the class. The instructor's immediate concerns range from the availability of the network-based presentation materials, the ability to interact with both in-class and remote *synchronous* students, and the ability to capture class materials and exchanges to enable *asynchronous* students to access the material at a later time. Neither category is really interested in, nor has time, to worry about the underlying issues of network availability, storage, throughput guarantees, and so on. The latter are expected to be provided by the infrastructures and models that support NBE systems in a seamless and unobtrusive way, and enable the users to go about their workflows of learning (students) and teaching (instructors) without having to give the NC "appliances" more thought than they would to a pen, paper, chalkboard, or an overhead projector.

An example of a more general environment for NC is the World Wide Web (WWW) [10, 9]. It could be the immediate platform for the above-mentioned NBE system (e.g., Web Lecture System [109]). The supporting infrastructure is the Internet which can be thought of as the global computer network consisting of computers running the Transmission Control Protocol/Internet Protocol (TCP/IP). The WWW is the subset of the Internet that supports the Hypertext Transfer Protocol (HTTP). The WWW is almost always identified with the HTML-based content to which it enables access. Beyond this low-level description, however, what makes the WWW worth using and studying is the presence of the material that WWW supports.

Fundamentally, what makes network computing worthwhile in an engineering sense is the combination of

- the need for some resource-intensive "killer" applications, along with the opportunity for developing those applications in a network-centric manner to share the cost and maximize the utilization of available resources

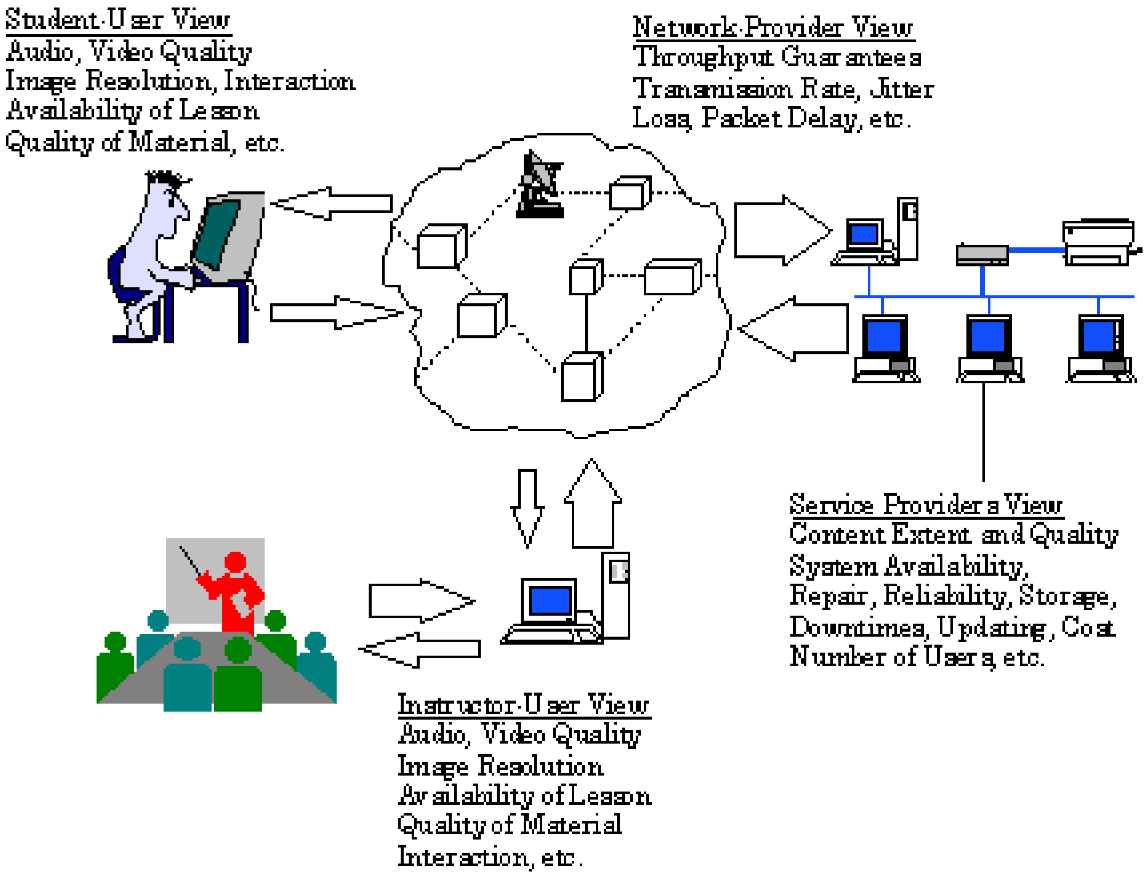

Figure 1: Perspectives on a Network Computing Architecture

- the need to provide less demanding, but guaranteed services, to thousands and millions of users by thousands of providers separated in both time and space.

This article surveys network computing in the large, as the intellectual discipline that underlies the approaches that address both the grand and everyday applications of today. There has been some industry interest in the concept of *network computers*, machines that may lack a local disk and rely on the network for functioning at all. This is to be contrasted with the notion of *national data and computing grids*, which promise to help ameliorate the resource crunch that is facing the nation and the Internet users. We view the former as one specialized infrastructure and architecture for network computing and shall address it only at a high level here, while the latter is more representative of the general network computing paradigm.

1.2 A Brief History of NC

Computing technology has evolved enormously over the past few decades. In our view, the evolution of the paradigms and architectures, and technology and infrastructure has tracked the evolution in applications. Each successive generation of technology has sought to remove the bottlenecks that prevent the expansion of the previous generation.

Network computing, as a concept, has been present since the early days of computing—in the form of distributed terminals serviced by centralized computing facilities, then through the workstation, client-server, and peer-to-peer paradigms, and nowadays through a combination of these and concepts such as the virtual local area networks (LANs), wide-area "compute-girds," the WWW, and so on.

generation	*technology*	*example*	*data*	*proc*	*UI*	*maint*	*NW Mbps*
central	main	DP	S	S	S	S	0
timeshare	main, phone	DP	S	S	C	S	0.1
C-S	mini, LAN	DBMS	C/S	C/S	C	C/S	10
Peer	workstn, LAN	CSCW	L/R	L/R	L	L/R	100
HPC	main, workstn, OC3	OLAP	L/R	L/R	L	L/R	$\geq$ 100
Limited NC	workstn, ATM	Office WF	R	L/R	L	R	$\geq$ 100
Open NC	workstn, Internet	VE, EC	R	L	L	L	0.1-100

Table 1: The Generations of Computing Technology

Table 1 summarizes the key features associated with each generation of computing technology: centralized, timeshare, client-server, peer-to-peer, high performance computing, limited NC, and open NC. The corresponding representative examples are file-based data processing (DP), database management systems (DBMS), computer supported cooperative work (CSCW), office workflows (WF), online analytical processing (OLAP), virtual enterprises (VE), and electronic commerce (EC). The data, processing, UI, and maintenance columns are about the location of the corresponding functionality or component. C, S, L, and R refer to client, server, local, and remote, respectively.

When computers were first networked in the sixties, they were expensive mainframes. The paradigm of usage was time-share in which multiple users could access a single mainframe. All of the computation was performed on the mainframe; the users communicated through plain terminals. With the development of database technology and minicomputers and the expansion of computing into newer applications, this paradigm was preserved although the size of the network grew and the type of end-user stations changed.

The early eighties saw the emergence of the *client-server* paradigm, which is a form of weak distributed computing. In this paradigm, the user sits on a computer, which handles the user interface and other application-specific computation. The computer could be a genuine workstation-type computer, PC, or an X-station. The client, however, relies on the server for all of its data management. The server can be a simple file server or a database server. The latter also provides advanced functionality such as

- concurrency control—to keep the various clients from inadvertently destroying each other's work
- transactional support—to guarantee atomicity of the work performed by a client
- backup and recovery—in case of media or system failure

The client-server architecture can be thought of as based on the remote-procedure call (RPC) mechanism. RPC provides a programmatic interface through which computing resources at remote hosts can be accessed. However, its fundamental shortcoming is that it is synchronous or blocking—in other words, the caller waits until the callee can produce a result. Thus it is not an efficient means to use computational resources.

The next generation of computing, already being deployed, is true distributed computing. This is also called the *peer-to-peer* paradigm. In this paradigm, the different nodes exchange asynchronous messages, in which neither party waits unnecessarily.

Interestingly, the WWW in its present incarnation follows the client-server paradigm. This is because HTTP, the protocol that underlies the WWW is a client-server protocol. A browser (working as client) can request information from a WWW server; the server cannot unilaterally send updates to the browser. The next version of HTTP will, however, admit both the present *pull-based* paradigm and the peer-to-peer *push-based* paradigm.

1.3 Scope and Organization

Over the years, the focal solutions were driven by the technological and usage bottlenecks. Network bottlenecks tend to put the computing power on the desktop, central processing unit (CPU) bottlenecks tend to provide distributed access to powerful centralized resources, while information distribution tends to promote peer-to-peer interactions. The main result of this evolution is the conclusion that, in general, as the complexities of the social and physical world in which computers are deployed are better addressed, it turns out that network computing can provide greater efficiency—and hence reduced cost and improved quality—in the construction and maintenance of complex computational applications.

Consequently, we base our presentation on a workflow-based treatment of network computing. Workflows are a series of structured human tasks, computations and interactions among them that arise during accomplishment of any set of focused activities (e.g, teaching, scientific problem-solving). In other words, a *workflow* is a multitask activity that consists of human and computational tasks coordinated in an appropriate manner. Workflows are often executed on a number of computer systems and over a number of databases. Nowadays, it is common to associate such systems with agents. Although *agents* are a common buzzword, all too often, agents may be nothing else but daemons, or specialized utility processes and programs that help NC systems operate in a more integrated and autonomous fashion. In general, agents are persistent entities with some autonomy, which perceive, reason, act, and communicate. The agent metaphor is a powerful one, but only when coupled with the metaphor of interaction. When we couple the two, we obtain *multiagent systems*. NC is fundamentally based on the notion of interaction. For this reason, we believe that NC environments of the future will almost certainly be populated by hosts of agents of the latter kind. In that context, four important layers of consideration are:

- *NC applications* which refer to the end-user systems that may be thought of as providing direct support in response to user workflow needs.
- *NC paradigms* and *architectures* are ways of thinking about how user workflow needs may be provided in distributed computational systems. They provide the means and methodologies for the global design of systems that realize user applications.
- *NC technologies* are system-level building blocks with which the global designs are instantiated.
- *NC infrastructures* provide the network and computing system functionalities on which the above technologies subsist.

The above concepts are discussed in more detail in the sections that follow.

2 Applications

Network computing has a large number of important applications. These applications, though in widely different domains, share the characteristics that they are often large-scale, distributed, heterogeneous (both in their software and hardware), open (in that they can grow and shrink dynamically), and composed of locally autonomous units. We provide an overview of some of the most interesting and important problem domains in which NC is applied.

2.1 Enterprise Integration

Enterprise integration is one of the oldest applications of NC, and in fact predates the emergence of the WWW by several years. This application arose as a consequence of the success, but also the limitations, of previous data processing technology. As computing systems came more powerful, databases became ever more prevalent in all kinds of organizations. For our purposes, *enterprises* are government or business organizations viewed in terms of the information they acquire, own, and manage in pursuit of their key functions. Enterprises became to depend on online databases for almost all of their business needs. All too often, several databases were built in an *ad hoc* manner in different parts of an organization. These databases, although individually useful, would not and could not talk to each other. Not only could the knowledge of an enterprise not be put together, it was often also the case that the different databases were mutually inconsistent. Naturally enough, this led to poor decision-support leading to losses in productivity, quality and, ultimately, in the bottom line.

In a conceptual sense, integrating preexisting systems is in general a harder problem than designing distributed systems afresh. Many systems, especially those based on older mainframe architectures, allow data to be accessed only through narrow, inflexible interfaces. Such systems are frequently termed *legacy* systems—apparently, the term was chosen with a disparaging connotation. Legacy systems and their interfaces cannot be easily modified. No reasonable approach would try to modify them significantly. This is because of two main reasons: (1) the complexity of the programming effort that would be required to achieve any modifications, and (2) the constraint that older applications continue to run as before, since they typically have a wide user base that relies heavily upon them. Thus, the integration must permit newly developed applications to coexist with previous applications.

However, integrating preexisting systems has major advantages over implementing distributed systems from scratch:

- The existing systems typically serve important business functions, which it would be undesirable to disrupt.

- The need for the integration results from the growth or restructuring of an enterprise, e.g., because of mergers and acquisitions, or because of a new way of understanding the information processes carried out in the enterprise, e.g., because of reengineering or refocusing. This need is not of the sort that can be addressed once and for all, but rather will naturally emerge again and again over time.

- The integration offers an opportunity for preserving whatever we can of the autonomy of the individual components. A redesign that results in a new conceptually monolithic system, even if improved, would not be quite easy to reuse. This is because monolithicity is a liability whenever a system needs to be modified. Modifications may be essential as needs within the components change, or another restructuring takes places, e.g., to separate out the business divisions that had been put together before.

As a concrete example, on which one of the authors previously worked [98], consider a large company that provides a variety of telecommunication services. We studied the workflow for establishing a telecommunication link between two specified points (this is termed service provisioning). In the original workflow, a set of paper forms was received that gave a number of relevant details about the service being ordered. These forms were entered into the system. A test was then performed to determine if the telecommunications equipment required for the order was already in place. If so, the service could be provided relatively quickly; otherwise, the processing would be delayed until the equipment was added.

In the original workflow, service provisioning typically would take several weeks and require coordination among many operation-support systems. Configuring the operation-support systems to perform such a task often would take several months. Our goals were to reduce this time to less than two hours and to provide a way in which new services could be introduced more easily. Our strategy for accomplishing these goals was to (1) interconnect and interoperate among the previously independent systems, (2) replace serial operations by parallel ones by making appropriate use of relaxed transaction processing [16, 35], and (3) automate previously manual operations, thereby reducing the incidence of errors and delays.

The reduction in time is of competitive significance in the modern business environment. The US telecommunications has gone through a phase of deregulation, which reduces or abolishes several governmental regulations, but also enables new companies to compete in markets that were previously controlled by government-supported monopolies. In the case of the enterprise we are discussing, a majority of its competitors emerged relatively recently. As a consequence, the competitors typically had newer computing systems, which were better optimized for the specific task at hand. As remarked above, this advantage of the competitors is only temporary—as each matures and as regulations and other operating conditions change, each of these would inevitably end up with a heterogeneous mix of legacy systems.

2.2 Virtual Enterprises and Manufacturing

Virtual enterprises (VEs) are a concept that resembles enterprise integration, but goes beyond it in terms of allowing multiple independent enterprises to come together, sometimes for only the duration of a single collaborative project [105]. For the duration of the project, the different participants in a VE behave as if they were part of the same enterprise, although they might preserve their autonomy and privacy with greater intensity than in the traditional case. VEs sometimes emerge from the business interests of the participating entities who mutually decide to form the given VE. At other times, e.g., in large government-funded projects, a VE may be formed by the funders by distilling and combining the best expertise of the individual bidders. VEs thus emphasize the challenges of openness and dynamism, and in the face of maintaining the autonomy of the collaborating organizations.

In principle, virtual enterprises can arise in any domain. However, they are often associated with manufacturing, because that domain lends itself well to collaboration among autonomous organizations. A representative effort is SMART, which is a multiyear, multimillion dollar manufacturing applications project being sponsored by the US government [99]. SMART stands for MES-Adaptable Replicable Technology. MES stands for Manufacturing Execution Systems, which describes the major kind of flexible manufacturing systems. SMART is being conducted by the US National Industrial Information Infrastructures Protocols (NIIIP) consortium [45]. SMART uses intelligent agents to control manufacturing execution [42]. A *virtual private network (VPN)* is a network that is restricted to the members of a virtual enterprise. One of the interesting SMART technologies is based on *information contracts*, which apply among the participating enterprises within a VPN. Such contracts are key in having autonomous entities interoperate for flexible manufacturing.

2.3 Electronic Commerce

Electronic commerce (EC) roughly corresponds to doing business through electronic or network-centric means. EC has been around as a concept since the early days of computing. However, earlier efforts at EC, which achieved only moderate success, were limited in their goals. What they primarily offered was some standardized means of exchanging information: such standards are known as *electronic data interchange* or EDI standards. EDI standards typically did not achieve industry-wide acceptance and were often just enforced by large vendors or suppliers on other corporations that sought to do business with them. Also, by and large, EDI and related EC work did not address commerce or trade as such, but only sought to facilitate the flow of information once the details had been negotiated and agreed upon in advance.

The modern vision of EC is far more sophisticated and ambitious. It is now considered within our reach to have automated means of conducting trade in an open setting. Service providers and users can be thought of as agents that can negotiate with another, and who can represent and reason about contractual details, and commit themselves, i.e., be empowered to commit the human whose interest governs their actions.

2.4 Computer Supported Cooperative Work

Computer Supported Cooperative Work (CSCW) is a class of applications that involve humans to collaborate in performing some task over the computer. CSCW involves tools through which humans can achieve the effect of a shared workspace even when physically distributed. Natural and important applications of CSCW are authoring—of programs or documents.

Some issues in CSCW are purely of a distributed computing nature, for example, to ensure that the documents being produced can be consistently viewed at different sites. Other work concerns techniques for version management and for assisting humans in merging divergent versions into a single consistent one—this task can in general not be fully automated, because merging two versions of documents requires intimate understanding of their content.

Another class of issues deals with explicit shared workspaces, not only those indirectly reflected in the artifacts being manipulated. This requires more sophisticated techniques of user interfaces and the creation of realistic or believable environments, for example, for virtual reality.

2.5 Scientific Computing

Like all other NC environments, modern scientific computing and problem-solving environments (PSEs) are collections of cooperating programs, tools, clients, and intelligent agents [38]. These components are integrated into an environment that facilitates user interaction (such as problem statement and solution engineering) and cooperative execution of the components charged with the solution tasks. An example is a system that would help an environmental scientist or regulator to pose environmental engineering questions (problems), develop, execute and validate solutions, analyze results (e.g., including coupling with Geographical Information System data and visualizations), and arrive at a decision (e.g., a cost-effective strategy to control chemical emissions). Such a PSE would consist of a management, analysis and computational framework that would be populated with a variety of numerical models and data that describe the science behind the phenomena, the solutions of interest and the decision rules [31]. It is usually assumed that a modern PSE is distributed across a number of central processing units that may or may not reside in one physical computer. In fact, the advent of high-performance computing engines and networks, the potential of new technologies to guarantee *quality of service (QoS)*, and the ready access to network-based information through the WWW is bringing serious numerical and problem-solving applications closer to a broad base of potential users. The next generation of large-scale PSEs is expected to operate in national '*compute-girds*: a mesh of high-performance computation, data storage and database nodes interconnected by a high speed backbone (2.4–10 Gbits/second) [4, 56].

Users expect not only the provisioning of high quality numerical computing algorithms and software, but also integration of these solutions with advanced computational and networking frameworks, and with day-to-day operational environments and workflows. The workflow paradigm enables appropriate description and analysis of coexistence and melding of scientific workflows with other workflows into which they have to fit, and of certain quality constraints dictated by these larger workflows. By describing scientific computing and problem solving as workflows, we allow for the advanced techniques being developed in workflows research, as well as performance and QoS work deriving from the high-performance networking and communications arena. These include sophisticated notions of workflow specification and of toolkits and environments for describing and managing workflows, and recognition of performance issues that have previously been reserved for analysis of networks (e.g., that end-to-end throughput observed by a user at the application level will typically be that of the device or process with the lowest throughput capacity in the path). In this way, NC supported scientific workflows are to problem-solving environments what business workflows are to NC-based enterprise integration.

2.6 Distance Education

Network-Based Education (NBE) refers to provision of education using NC resources. We take a system view of such education using the workflow concept [91, 108, 107]. This concept recognizes the educational process as a system which involves interactions among a variety of individuals including teachers, researchers, learners, advisors, and administrators; through a series of workflows primarily involving the access, creation, teaching, or manipulation of the subject matter. These activities can become particularly intense and difficult to manage and synchronize. Understanding the educational workflows is key to effective application of technology to education. Only when advanced computer technology is correctly mapped to the educational process through the workflow model, can its fundamental benefits begin to approach full realization.

The most important NBE system entity, and the principal quality driver and constraining influence is, of course, the user. NBE users can be classified into a number of categories. Four nonexclusive important categories are students, instructors, authors, and system developers [11, 107]. Examples of other important general categories of users are parents of the students, employers of continuing and adult education students, and educational administrators. Special categories of special interest are K-12 users, community college users, university users, and adult education users. Functional and usability requirements derive, for the most part, directly from the NBE user profile.

System developers are responsible for the development and maintenance of the system software and resources, authoring tools, courseware tools, and so on. Authors are courseware developers. It is essential that authors be pedagogical and content experts, but not that they be system experts. Instructors deliver the course material. They sample and combine existing lessons, customize, and update and develop courses and projects. System support for tutoring, student-instructor interaction, and student evaluation is essential. Students are the most important users of the system. They require appropriately reliable and timely lesson delivery, easy-to-use interfaces, collaborative support in local and remote joint projects, instructor's help, and so on. Since the class of the future is likely to include students and instructors who are widely separated geographically, who may not able to attend lectures on a preset schedule, and who come with different backgrounds, the tasks for system support and instruction should scale the barriers of space and time as well as of student diversity. Therefore, a successful large-scale wide-area NBE system should:

- support a large number of students that range from naive to sophisticated.
- support construction and delivery of curricula to these students by facilitating the work of thousands of instructors, teachers, professors and parents that serve the students.
- support generation of adequate content diversity, quality and range. This may require support for many hundreds of authors.
- be maintainable with a relatively small number of systems personnel.

2.7 Open Information Environments

Open information environments were indirectly discussed in some of the applications above. They are a unified way of thinking about environments in which information resources may be added or removed dynamically. Thus the information resources are autonomous—a property they inherit from their human owners. Current environments based on the WWW are usually open in the above sense. This is what makes it difficult to enforce any consistency on the behavior of specific applications. A consequence of openness is that standards emerge slowly and are typically "least common denominator" in their extent. Thus on the WWW, HTTP, and HTML are standardized, but the content or even additional structure of the documents served on the WWW are not. A number of other efforts and standards either exist, or are in the making (e.g., Open GL).

Open information environments require a wider variety of information access techniques. Traditional structured or text databases support querying, where a query is formulated by the user, and its evaluation yields some results that are returned by the database. We often have sources that produce information continually in a stream of "articles." Such information streams must be filtered, by specifying some kind of a pattern that selects the desired articles.

A major issue in open environments is *resource discovery* because, unlike in closed environments, it is not clear where a query or filtering request may be directed [88, 13]. Whereas in retrieval, the consistency of the results is often an issue, in filtering and resource discovery, consistency is not well-defined, but the relevance of the results is crucial. Retrieval and filtering are of about the same complexity in closed and open environments, but discovery is significantly more complex in open environments. To be scalable and manageable, it requires some kind of distributed indexing techniques [28].

2.8 Ubiquitous Computing

Ubiquitous computing refers to the vision—fast becoming a reality—that computing should be accessible to users no matter where they are [52, 59, 82]. One such item are *wearable computers*, continuously active computers that, in the future, are expected to inhabit our wallets, briefcases, clothes, and possibly bodies (e.g., communication implants, health-monitoring implants, small information appliances with powerful functionality) [64]. Ubiquitous computing thus typically requires mobile computers as well as infrastructure for mobile telecommunications, such as cellular telephony. While cellular telephony is not yet available everywhere on earth, several major efforts are underway in populating our skies with networks of satellites to enhance its availability in the near future. Wireless networking has not yet been standardized to the extent that one may use a single phone or wireless data-line even across the regions where some cellular telephone infrastructure exists, but that day is coming.

3 Paradigms and Architectures

3.1 Distributed Objects

An NC environment provides all the computational facilities necessary to solve a target class of queries, interactions or problems efficiently. A natural unit for interaction is an *object*. The advent of wide-area high-performance networks has prompted development of distributed object-oriented environments that make use of geographically widely separated computing resources, and allow an equally distributed community of users. A typical distributed object environment is a combination of distributed user interfaces, agents, object libraries, knowledge bases and a variety of enabling and integrating technologies that facilitate user interaction (such as problem statement and solution engineering) and cooperative execution of the components charged with the solution tasks, e.g., [38, 81].

This need for effective and efficient communication among the NCE components (or objects) is recognized by both researchers and software manufacturers. In recent years, this has resulted in a proliferation of communication building blocks, or *middleware*, for distributed scientific computing and problem solving. The functionality and the mode of operation of middleware usually introduces extra overhead. This overhead is accentuated in environments where the NC resources (such as workstations and supercomputers) are interconnected by high-speed links (e.g., switched 100 or 155 Mbps links). Such an environment can magnify any middleware communication deficiencies and make the middleware a major performance bottleneck. Some specific examples of the middleware (e.g., CORBA, DCOM, PVM, MPI), and its performance are discussed below.

A higher-level issue associated with distributed objects is in the management of their interfaces. A number of approaches, based loosely on the notion of directories—white pages and yellow pages—exist. A *type broker* is an object, listening at a well-known address, which carries information about the "types" supported by other objects. These types are not just the data types but include the signatures of the different functions or services supported by those objects. An application can use a broker to find the objects that offer services of a desired type and then communicate with them directly.

3.2 Collaborative Environments

Students taking classes at a distance need to interact with each other and with the instructor, distributed groups of scientists and engineers that work on a problem or a report need to communicate with each other and show their work, geographically distributed physicians may wish to consult with each other regarding a patient. All these tasks can be made easier and more productive through use of NC. In general, network-based collaboration among, and with, users of NC systems can significantly enhance their experiences and improve their joint work results. This requires groupware that facilitates file sharing, collaboration, discussion, and so on.

For example, in the area of NBE, PLATO and NovaNET [11] were the first multimedia learning environments that supported extensive interaction among students as well as communication between the tutors and the students through a facility that lets one or more of the collaborators watch and interact with the screen of another collaborator. There is currently a host of commercial tools that provide similar or more extensive facilities. These range from teleconferencing, to white-board sharing to chat-rooms. Examples are full versions of graphical WWW browsers, the MBONE toolset [65], numerous Video-over-IP ventures, Microsoft's Netmeeting, and so on. In general, groupware for collaborative project development needs to consider both synchronous and asynchronous interactions, group document control and maintenance. A related, but more encompassing, concept is that of *collaboratory* [89, 84, 79]. Collaboratories provide virtual spaces that enable collaboration among parties that are not present at the same time or place. Interaction groups form as needed and group members share their documents, software, data, instruments and knowledge in a virtual environment that is an extension of their natural work environment.

3.3 Multidatabases and Interoperable Systems

The conceptual schema of a database describes the structure and content of the information stored in a database independent of its storage in physical memory [7].

Traditional databases keep all of their data at one site. This has obvious problems of access time and reliability. Accordingly distribution of some kind is desirable. The simplest form of distribution is by replication in which the different sites have the same schema. A variant is one which the information is partitioned systematically to improve local processing and reduce redundancy. For example, each branch of a bank may store only its local information. However, the schemas are the same everywhere. A more general approach allows differing schemas, but the schemas are still built from the same vocabulary and are designed to combine properly with each other. The local schemas are merely projections of the global schema, which provides the sole way of accessing the database. The location and replication of the information is hidden from the user. The above kinds of systems fall under the category of *distributed databases*.

More general and powerful approaches are obtained when the sites have different schemas, and the schemas were not designed at the same time. Such approaches are heterogeneous in their schemas. Heterogeneity is typically not added by design, but arises because of a realization that the information already stored in the various databases ought to be shared. New application programs are written that access information from the different databases. However, the previously designed programs that need information from a single site continue to function as before. This is called a *federated* database or a *multidatabase*.

Interoperable systems are based on the ideas that the components and their schemas are entirely autonomous. No global schema is required, although middleware would be required to make the already functioning components interoperate. The data may be accessed through multiple applications and languages.

3.4 Transaction Processing

We think of transactions as an abstraction for programming composite activities in information environments.

3.4.1 Traditional Transactions

A traditional transaction is a set of operations on a database that satisfies the *ACID* properties. That is, a traditional transaction must *a*tomic (all or none of the operations happen), *c*onsistent (concurrent execution of the transactions do not violate any integrity properties of the database), *i*solated (concurrently executing transactions do not share any data, and *d*urable (once a transaction commits, its results are permanent, unless updated by another transaction). ACID transactions have proved remarkably effective in a number of data processing applications [46]. Unfortunately, they are not well-suited to heterogeneous systems, which are a large and important component of the environments for network computing. There are two main reasons.

- Atomic commit protocols are inefficient because of distribution and often impossible because of autonomous legacy applications. Local autonomy is often sacrosanct, because of technical and political reasons. Some resources, e.g., legacy databases, are technically closed in that they have no visible precommit state. A visible precommit state is essential for executing two-phase commit and other mutual commit protocols involving those resources. Further, some components of heterogeneous systems are owned and managed by different organizations or divisions of an enterprise, which cannot or would not grant control to another agency.

- The semantic requirements in heterogeneous applications are often complex and need more sophisticated task structuring than ACID transactions [35].

3.4.2 Extended Transactions

For the above reasons, a number of extended transaction models (ETMs) have been proposed [35], which generalize the ACID model in different ways. Each ETM requires customized scheduling.

Let's consider the DOM model as an example extended transactions model [15]. This model considers a composite activity as a *multitransaction*, which consists of a number of component activities. The activities may be mutually temporally ordered; some may be optional. Some activities are designated *vital*, which means that their failure causes the failure of the entire multitransaction. Activities can have dependencies among them to indicate the steps necessary when different combinations of the activities fail or succeed.

3.5 Workflow Management

3.5.1 Modeling

Recall that workflows are composite activities that achieve interoperation of a variety of system and human tasks. Workflows must satisfy subtle domain-specific integrity and organizational requirements. Consequently, flexibility in execution is crucial. A promising means to achieve flexibility is through declarative specifications with automatic distributed scheduling techniques.

For traditional (homogeneous or centralized) environments, database transactions provide effective and robust support for building applications. Unfortunately, corresponding support is not available in heterogeneous environments. The application programmer must procedurally encode all necessary semantic requirements. Workflows are widely regarded as the appropriate concept for structuring complex activities, and workflow management systems would provide functions analogous to those provided by present-generation transaction monitors. Accordingly, increasing attention has focused on workflows [29, 49].

As the prevalence of heterogeneity is being appreciated, the importance of workflows is increasing. Although scores of workflow products exist, relatively few of these are integrated with databases. Even the best of those integrated with databases often have centralized implementations, and offer little support for semantic or recoverability properties. For example, XSoft's InConcert operates on a single server—it lists distribution as a future challenge [86]. Action Technology's ActionWorkflow is also centralized, and more geared toward computer-supported collaborative work than database transactions [117]. IBM's Flowmark offers strong support for business processes, but has a centralized implementation. However, associated with these products are useful techniques for process modeling and capture. These include ActionWorkflow's "language/action" model of human interaction, and Flowmark's activity-network model of processes [60].

3.5.2 Programming and Enactment

Workflows are not only modeled but must also be enacted. There is much anecdotal evidence that the best way to program workflows (or any kind of software models) is to make the models themselves executable. Indeed, this is what is done in all the workflow systems and prototypes.

Procedural Approaches The traditional approaches to programming and enacting workflows are procedural. The workflows are specified essentially as activity graphs. Executable code is generated from these graphs in a straightforward manner. Hooks are added in for whatever external procedures or applications must be invoked. The implementer has to do a fair amount of system integration work in ensuring that the databases are accessed properly and the graphical user interfaces reflect the ongoing state of the workflow.

Declarative Approaches A newer class of approaches is declarative. It is motivated by the fact that activity graphs by themselves give no idea of the structural properties of the workflow, but higher-level abstractions to capture the interrelationships among the composite activities must be captured.

The simpler abstractions involving control and data flow can be borrowed from the ETM literature. However, the sheer variety of ETMs precludes hardwired approaches for scheduling each ETM. The declarative approaches provide a small number of primitives that can be used to specify ETMs [41, 24, 2].

The success of the declarative approaches depends on whether they include a language that is simple yet expressive, and whether specifications stated in that language can be automatically processed. The pioneering work of Chrysanthis & Ramamritham developed a simple temporal language that could capture the scheduling aspects of all of the known ETMs. However, they didn't study the scheduling aspects *per se*. Work by Attie *et al.* [2] and Singh [91] has addressed the latter challenge as well.

3.6 Agents and Multiagent Systems

The term *agent* is overloaded in the literature. For our purposes, agents may be defined as persistent computations with some autonomy, and which perceive, reason, act, and communicate. Agents can capture or serve the interests of human users, information sources, or service providers. These agents may be independently created by different parties, or may even be preexisting. For this reason, agents can be naturally applied in NC.

The simplest applications of agents are in simplistic user interfaces, as programs that filter electronic mail, or those that perform network searches. Such applications of agents do not rely on many of the properties that make agents scientifically interesting, and often could be more succinctly described in terms of ordinary programs.

The agent metaphor is the most valuable when coupled with the metaphor of interaction among computations. Their combination yields *multiagent systems*. Multiagent systems promise a natural, compositional way of building complex NC systems [51]. The interactions specify how the computations of the agents come together to obtain the desired behavior. In contrast with a single-agent system, in a multiagent system, the agents may cooperate to search different components of the information space and share their findings. Conversely, the agents may compete or negotiate as in electronic commerce or even to decide how to split the information space that they are supposed to search.

Multiagent systems also promise the development of compositional systems from autonomous, heterogeneous, but interoperating parts. This is because the agents can be locus of autonomy; they can satisfy the minimum standards for interaction while hiding the implementation details of the systems they represent in a multiagent system. The need for interoperability is driving a lot of research into languages and techniques for

- agent management, by which agents can register themselves with some sort of a domain name server, advertise their capabilities, be found, and have requests made of them.

- agent communication, by which agents may send meaningful messages to other agents to inform them of various facts, to request services, or to promise services of their own.
- ontologies, by which the agents can share the terminology in which their assertions, requests, and promises are expressed.
- interaction protocols, by which agents may participate in certain routinized interactions such as in a specific auction protocol.

A *de facto* standard for multiagent system architecture is emerging. This architecture includes roles for agents to represent human users, information resources, brokers, and ontologies. We return to this below.

3.7 Enhanced Workflow Management

Section 3.5 describes workflow technology as it exists today. Although much progress is being made on workflow technology, some important issues remain unaddressed. These include (1) semantic or recoverability properties demanded by serious applications; (2) distributed implementations; (3) ease of specifying and modifying workflows; and (4) formal models of workflow computations to give a solid foundation for the reasoning necessary to design, analyze, and reliably enact workflows. This is not to blame the products—the above properties that are not handled by the products remain a research challenge today [54]. Indeed, it is their success in important applications, e.g., [67], that fuels a continued interest in research into workflows.

To remove these limitations requires extensions along at least two dimensions. We envision workflows as incorporating both semantic and organizational requirements. The former category includes abstractions from extended transaction models that relate to correctness, concurrency control, failure handling, and recovery. The latter category includes abstractions from business modeling that relate to organizational structure, roles of participants, and coordination among them and the information system. These topics are being actively researched, e.g., [18, 78]. However, it is significant that these efforts would be greatly facilitated by an infrastructure that includes a generic and rigorous approach to workflow specification and scheduling. For its intended use, the specification and scheduling functionality should apply to activities whose component tasks are not restricted in any way, our primary interest is in workflows with a databases and information systems focus. Some of the constituent tasks may be database transactions; some may involve human interaction; some may be other kinds of computations.

Enhanced workflow management is a natural application of agent techniques. Agents can provide the flexibility and high-level interactions that it requires. Further agents can help us think not of isolated workflows, but workflows that interact with other workflows, *horizontally* and *vertically*. We discuss these concepts after in the context of the education example introduced in Section 2.6.

Horizontal Integration of the Flows Education workflows are expected to coexist, cooperate and meld with other user workflows. Therefore, they must support compatible interfaces and constraints. We call this *horizontal* integration of the workflows at the level of end-users. For example, many students from industry that work during the day may prefer to incorporate the majority of their continuing education into their daily or weekly routine at times that suit them, e.g., evenings or weekends, because they cannot match their work-place processes with the traditional school or college teaching workflows. However, this particular challenge to traditional education workflows cannot be met without extensive technological and pedagogical support which allows (1) decomposition of the synchronous teaching/learning cycle into a primarily asynchronous component (with minor synchronous interactions), and (2) at the same time preserves and maximizes the quality of learning and the knowledge transfer rate that is normally associated with the classical synchronous teacher-student interaction. Other functionalities are needed in the case of other types of horizontal integration. This frequently requires strong support for collaborative activities.

Vertical Integration of the Flows Interactions and negotiations also have to take place between the end-user layer of an NBE environment and the underlying infrastructure (platforms, software, computer hardware, interconnecting networks) in order to provide the throughput, keystroke delays, jitter, and other services, that an NBE application or user expects. We call this *vertical* integration of education workflows with event, control and data flows that occur at infrastructure layers. The network- and platform-related flows and QoS capabilities of the information infrastructure (e.g., power of the user platform, network capacity, supercomputing facilities) have to be appropriately matched and interfaced with the needs of the user's educational and training workflows.

Applying workflow technology to a specific university course requires information about the syllabus, participants (both faculty and students), schedules, and instructional facilities and technology, and development of the corresponding *operational profile*. An operational profile is the set of relative frequencies that tell us how often a particular function or capability is requested in practice [68]. Specifically, given a syllabus, schedule, and the student profiles, one would first categorize the students by qualifications and learning styles, then produce a mapping between the syllabus topics and the student learning models. This would allow mapping of the needed content teaching approaches to content topics. This mapping may include the placement of feedback points, an estimate of the process feedback rates, location of testing points and reinforcement of learned material. The final step would be to map these needs to NBE system functionalities, based on instructor/author qualifications and preferences, available resources, etc., to obtain an operational profile that needs to be supported during the course. The mappings and the operational profile allow us to recognize teaching alternatives and introduce adaptive or fault-tolerant teaching into the educational model.

4 Technologies

4.1 Distributed Computing

Typically, an NC environment consists of a number of cooperating subsystems. Each of the subsystems can be a "mini-NC" consisting of modules that work together to solve a problem. This is more evident in an environment where the workload is distributed among a number of machines. In fact, there is a growing trend to view NC modules as (specialized) dynamic software parts or objects. For example, some can be repositories for high-level scalable computational models which can be used by other network-based PSE subsystems whenever necessary, others can be data centers, yet other can be actual (or virtual) computation engines, etc. In general, an NC subsystem might be expected to export its functions to other subsystems or allow direct user access to them through a graphical user interface (GUI). To accomplish this in an effective and efficient fashion, it is usually a good idea to use software "glue" (middleware) that has an established and tried performance and functionality. Effective NC presuppose some combination of the following functionalities.

Unicasts and multicasts An NC subsystem that requires the services or functions of other utility processes can obtain them by sending peer-to-peer messages requesting the service. For collaborative systems, multicast (or one-to-many) messages are useful for collecting status information, notifying about events, broadcasting common information, and so on.

Synchronous and asynchronous messages The communication mechanism should support synchronous messages that allow client processes to issue requests to NC components and to wait until they receive replies from them, and asynchronous messages that allow clients to issue messages and not wait for replies for them. An asynchronous mechanism is important for systems that use messages to trigger functions and to notify events.

Efficient data transfer Some NC subsystems, such as visualization or data-mining tools, may require transfers of large volumes of data between machines. It is important that the communication infrastructure be able to do this efficiently and without negative impact on NC system performance.

Interfacing with external processes NC components may need to communicate with other processes and systems that are not part of the NC infrastructure but whose services or support are needed to solve a problem. In general, these outside processes may communicate using a protocol, messaging and communication system that is not part of the NC environment. For example, a PSE subsystem may need to keep track of other processes on a machine to calculate system load. Its communication infrastructure should allow and support these external communication channels.

Remote access to services The utility of an NC may be greatly enhanced by allowing the functions of the individual subsystems to be invoked from other processes or subsystems. For example, such remote access to NC functions and methods allows an NC to be used as a component in a larger system or as an agent in a collaboration for another problem.

Quality of Service (QoS) The utility of an NC depends considerably on the quality of service that it can provide to its users. NC middleware should be QoS sensitive, and it should have functionalities which allow: (1) communication of application level QoS needs to lower levels; (2) communication of infrastructural QoS constraints to the application; and (3) ability to negotiate (on behalf of a user) a set of QoS parameters that maximize an application-level objective function. For example, for education, an the ideal QoS parameters would maximize knowledge transfer and retention. For environmental decision support, the general objective would be to minimize time to make correct decisions. More specific objective functions are needed for different system use modes.

4.2 System Integration

When (heterogeneous) modules have to interoperate in a single system, it is advisable to not have to modify the modules themselves. Similarly, reuse of standard functionalities from libraries of functions or objects should be preferred to reimplementation. In general, middleware is the answer to the challenge of reuse and easier user of lower-level functions. It is based on the idea that a small amount of “glue” may be inserted between mismatching, or more basic or more difficult to use, components in order to enable their interoperation. We restrict ourselves to information-level middleware, although middleware can in general arise even at the lower levels of the system architecture.

Some examples of middleware are numerical functions available through numerical libraries, graphical functions available through graphical libraries and reusable object libraries. Other examples are wrappers or mediators [113]. Wrappers perform simple language or protocol translations. For example, a language wrapper may map the operators in a text keyword search query into operators that would be recognized by the underlying system. Another language wrapper may map a structured database query into a lookup on specific fields in an HTML form. A protocol wrapper may map a remote procedure call into an asynchronous query for which the response will be matched to the query and returned to the calling program. Mediators are like wrappers, only more sophisticated. They can encode larger varieties of metadata, e.g., about the reliability of the information in the database or about how the values in the database map to values in another database. Mediators use encoded knowledge about data to create information for higher-level applications, providing a logical view of the underlying information. Mediators can, in principle, learn from the data. The popular generic agent-based architecture, shown in Figure 2, incorporates mediators.

A prominent example of NC communication middleware is Parallel Virtual Machine (PVM). A virtual machine is an abstract environment which is (ideally) platform-independent, and protects the programmer from the idiosyncrasies of the distributed computing systems on which the application will operate. Another example is Inferno, a product developed by AT&T. Of course, a popular virtual machine is offered by Java [61]. The general discussions that follow (including that on performance of middleware) apply to a broad range of middleware software, including Java.

4.3 Middleware

Of special interest to NC are two types of middleware that facilitate exchange of messages and information objects, respectively. It is important that this middleware not impose an overhead that would translate into failures to deliver the QoS expected by the users.

4.3.1 Message Passing

An important part of a modern NC framework is its ability to facilitate effective and efficient communication among the NC components (or objects). This is recognized by both researchers and software manufacturers, and, in recent years, it has resulted in a proliferation of communication building blocks for distributed computing. The best known examples are PVM and Message Passing Interface (MPI) [100, 39]. They are specialized message passing libraries for scientific and other computing that in combination with message and process brokers allow relatively easy distribution of a parallelized problem over a number of processing units in order to increase the system's computational performance. Computation platforms include everything from laptops running Windows 95/NT to multiprocessor supercomputers running Unix. Although not originally intended for this purpose, both PVM and MPI can be used to distribute not only fine-granularity solution elements (such as code segments) but whole programs and PSE parts.

PVM was originally developed by Oak Ridge National Laboratory and University of Tennessee [77, 39]. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine. It transparently handles all message routing, data conversion, and task scheduling across a network of computers with diverse hardware and operating systems. An application is cast as a collection of cooperating tasks, which access PVM resources through a library of standard interface routines. These routines

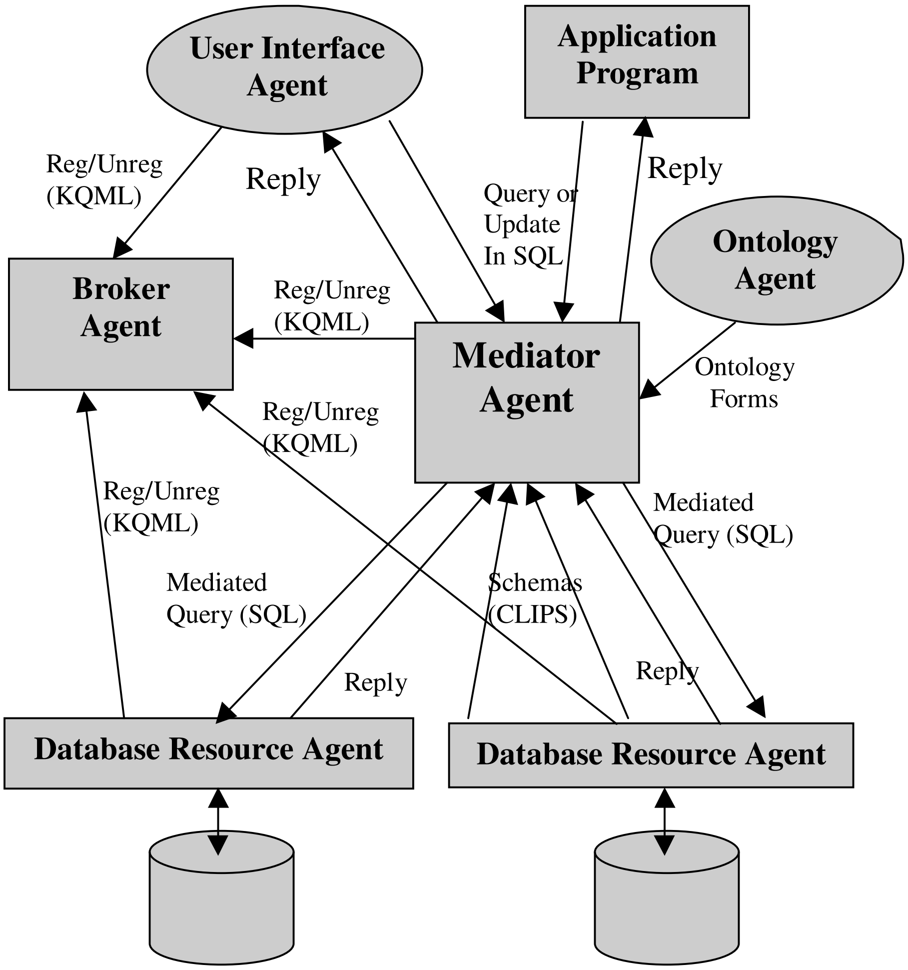

Figure 2: Perspectives on a Network Computing Architecture

allow the initiation and termination of tasks across the network, as well as communication and synchronization among them. PVM tasks may possess arbitrary control and dependency structures. XPVM provides a graphical interface to the PVM console commands and information, along with several animated views to monitor the execution of PVM programs. HARNESS is the next generation of this technology. It explores dynamic virtual machine capabilities, such as virtual machines collaboration, merging, or splitting, and plug-in interfaces for dynamic extensions to a parallel virtual machine, as well as methodologies for intelligent parallel applications, i.e., those that discover each other, dynamically attach, collaborate, and cleanly detach.

MPI is the outcome of a community effort to define the syntax and semantics of core message-passing library routines that are widely useful and efficiently implementable on a range of parallel processors [100]. MPI is not a complete and self-contained software infrastructure. It does not include process management, (virtual) machine configuration, support for input and output, and similar. MPI is limited to the message-passing (parallel) programming (MPP) model, which is a distributed memory model with explicit control parallelism. The message passing model is regarded as a viable means of programming both on multicomputers and on heterogeneous networks of workstations. This is considered a definite advantage when developing applications that may be ported afterwards onto a massively parallel multicomputer. Processes in MPP can only read and write into their respective local memory. They synchronize with one another by explicitly calling library procedures. Data is moved between local memories sending and receiving messages via explicit procedure or subroutine calls.

Other examples of similar software are P4, Express, and Linda. P4 is a library of macros and subroutines developed by Argonne National Laboratory for programming a variety of parallel machines, and support of both the shared-memory model and the distributed-memory model. Express is a commercial collection of concurrent computation tools. The idea is to start with a sequential program and following a recommended development lifecycle transform it into a parallel program using the provided toolset. An alternative to both shared memory and message passing parallel programming is provided by Linda. Linda is a concurrent model that uses a "tuple-space" abstraction for communication among cooperating processes [19, 20]. It is an abstraction of distributed shared memory with some important differences, including association, and is available as a commercial product.

4.3.2 Object Exchange

Another flavor of information exchange middleware is a variety of, usually CORBA-compliant, commercial object brokers that can be used to construct NC applications. The Common Object Request Broker Architecture (CORBA) was proposed by the Object Management Group (OMG) as a standard specification for distributed object computing [90]. CORBA makes possible the reuse of software through distributed object computing. The primary component of CORBA is the ORB core which provides transparent transfer of requests and replies between clients and object servers. CORBA defines two types of method invocations that a client can use to invoke an object's methods. The Static Invocation Interface (SII) is similar to a local method call where a client can invoke a method on an object without knowledge of the object's location or implementation. The Dynamic Invocation Interface (DII) is a more flexible approach than SII. Here, a client does not require compile time knowledge of the methods supported by an object. It can retrieve information about the object method definitions from a repository, and dynamically construct a method invocation request. CORBA toolkits developed by different manufacturers enable creation of frameworks composed of cooperating distributed objects. A client can invoke methods on objects residing in any process, on any machine. The distribution mechanism is kept hidden. Objects residing in a different process are managed by a server process, which receives method invocations directed to the objects that it manages and returns the results back to the client. CORBA implementations use a daemon process to manage server processes on a particular machine.

4.4 Itinerant and Stationary Agents

There are two main ways to implement agents: as *stationary* or *itinerant* (also termed *mobile*). Stationary agents are generally implemented using some kind of a rule-based system. They include functionality to access local information resources, e.g., through mediators. Most importantly, they communicate with other agents, e.g., to inform, request, or make promises.

Itinerant agents, by contrast, are designed to change their location while executing. There is a lot of interest in such agents, and they seem to capture the popular imagination. However, as a consequence of their mobility, such agents tend to be smaller than stationary agents. The sites on which they execute must be trusting or more secure so that incoming agents cannot damage them. Conversely, they must guarantee that the integrity of an incoming agent will not be violated.

We emphasize that the key difference between itinerant and stationary agents is not conceptual but implementational. One can achieve the same functionality with either approach. Itinerant agents demand stronger security from a system, but can potentially lead to superior performance in settings where the data is large and the customized programming is small. However, a convincing empirical case remains to be made.

4.4.1 Management

Agent management refers to the set of functionalities through which agent systems can be managed. These include a way for the agents to register and unregister themselves, advertise their services, find other agents, and transparently change their location if necessary. Although agent systems have long included such functionality, there is a recent push to standardize it that is being supported by the Foundation for Intelligent Physical Agents (FIPA) [37].

4.4.2 Communication Languages

If separately developed agents are to interact intelligently—whether to cooperate or compete—they must be able to communicate with one another. Successful communication presupposes a shared language. Some aspects of the language are in the application-specific terms used—this relates to data heterogeneity, and one of the best ways of addressing it is through the use of ontologies [73, 50].

The other major aspect of communication is in the actions that the agents perform through communications. This follows the key intuition of Austin that communication is a kind of action [3]. These actions are termed *performatives* and are classified into a few major categories: assertives (informing), directives (requesting or querying), commissives (promising), prohibitives, and declaratives (causing events, as in electing a leader process).

A shared language must have a shared syntax and semantics. Recently, a standard has emerged under the sponsorship of the FIPA [37], which seeks to include the previous DARPA sponsored work on the Knowledge Query and Manipulation Language [58]. This work reflects great progress, although an objectively verifiable or falsifiable semantics remains to be discovered [96].

5 Infrastructure

5.1 Computers and Operating Systems

The lower layer of infrastructure that supports NC includes the actual networks: the wires, light-pipes, wireless equipment, routers, switches, networking cards, firmware and software that operates these units. It also includes the physical computers and other network-related devices and appliances (e.g., input and output devices), that are needed for full operation of an NC application. The computers can range from laptops to supercomputers. Operating systems can be equally varied. Two special groups of computing devices are network-oriented and deserve a closer look.

5.1.1 Network Computers

Network computers are envisaged as diskless workstations that download all of their executable code from the network. The potential advantage of such machines is that their administration might be simpler. Managing a large number of networked computers, such as in a large corporation or university, is a major problem, one of whose aspects is upgrading the software on the various computers. Thus a potential solution is highly attractive. Network computers can be useful in restricted settings where the user may have no say about the software running on their machines. For example, all bank tellers would be required to use the same version of the account maintenance software, and people with dumb terminals may be willing to use whatever versions of software are installed on the servers.

However, the value of network computers to assist in this task is less clear if the users have any say about what software they run, and in what versions. In such a case, the problem of maintaining the computing environment will be multiplied back to its present size. Having different software on one site from where it can be downloaded is not an advantage any more, precisely because of the development of other aspects of network computing—if you can manage remote computers, not much is lost by letting the software remain there as well.

5.1.2 Mobile Computers

Although all computers can in principle be carried along, mobile computers are those that are conveniently portable by users. There are three main kinds of mobile computers:

- Laptops are the largest and most powerful variety. They can handle almost all of the tasks of a desktop computer. However, when mobile, they would typically be constrained by a lower bandwidth. Although laptops are ideal for certain computation-intensive tasks, such as document preparation or interfacing to complex scientific computations, they are not ideal for lighter tasks such as messaging or information access.

- PDAs or personal digital assistants are the smallest and lightest variety, which can be used for simple tasks such as keeping one's schedule, accessing email, or accessing information, such as maps, over the WWW [59]. With increased communications functionality, these are becoming what are now called Communicators.

- Wearable computers refer to a range of hardware [64]. Sometimes they are fairly large computers, comparable to laptops that are carried around in a back-pack. More appropriately, however, they are computers that can be carried attached to one's clothing—on a belt or arm-band. Wearable computers rely extensively on specialized interfaces the have for hands-free operation. Thus they can take input by speech of gesture, and output by audio or on displays that are also wearable, e.g., by projecting on the user's special eye glasses.

What makes mobile computers interesting are the applications they enable and the special requirements they have for networking, leading to special QoS challenges.

5.2 Networking Support

In general, a computer network is a system that connects end-user workstations and devices separated in space. A classification into local area networks (LANs), metropolitan area networks (MANs), and Wide Area Networks (WANs) is conventional. LANs connect computer systems that are close together, usually not more than few kilometers apart. WANs span countries and continents. MANs are usually not separately studied and are included in one of the other two categories.

A number of underlying technologies exist. LANs are often connected via Ethernet (10 Mbps), Token Ring (4–16 Mbps), ATM (25, 100, 155 Mbps), Fast and Gigabit Ethernet (100 Mbps, 1 Gbps), or FDDI (100 Mbps). The technologies for WANs tend to be even more varied. They range from T1 (1.544 Mbps), T3 (45 Mbps) to ATM over SONET (155 Mbps–10 Gbps). The hardware that actually directs and moves the packets and cells and frames ranges from switches, to routers to bridges.

5.2.1 Network Protocols

We can view the infrastructure that supports NC as a hierarchy of three layers: hardware and data transmission infrastructure (or data-link layer), networking and transport (or OSI Layer-3 protocols and OSI Layer-4 protocols), and applications. This is illustrated in Figure 3. The first layer varies widely and can include media access methods such as Ethernet, FDDI, ATM, and token ring.

The middle layer includes the so called TCP/IP suite. The Internet Protocol (IP) standard and its associated protocols (e.g, ICMP, ARP, RARP) are designed for delivering information through systems of packet-switched computer communication networks [76]. The basic blocks of data exchanged between end-to-end IP layers are called IP datagrams. Data sources and destinations are network hosts identified by fixed length addresses. The IP also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through small packet networks. IP provides a connection-less and unreliable service. Any association among datagrams, flow-control, and any error checking and reliability functions (beyond a basic self-check of its headers) needs to be provided by higher layers. The current version of IP, IPv4 is reaching the end of its useful life, primarily because of the limitations of its 32-bit address space. Soon, it will be replaced by a new protocol, IPv6 [14, 30].

Two examples of middle-layer protocols are TCP and UDP. TCP resides above IP layer. It is connection-oriented, and it assures reliable, full-duplex byte stream communication between user processes. Most Internet applications use TCP. User Datagram Protocol (UDP) is connection-less and unreliable, but has low overhead. For this reason, UDP is employed by a number of multimedia applications that provide their own reliability and pacing rules.

A recent, but prominent, member of the top layer is HTTP. It uses TCP, and it is the foundation protocol for WWW. HTTP is a transaction-oriented client/server protocol and is most often used between a WWW server and a WWW browser. An example of a more ancient, but still active, member of the top layer is

FTP for transmission of files between systems. There is a number of other applications at the same level, including Telnet, and a simple e-mail transfer protocol (SMTP).

Higher level network-based tools needed for collaborative components of the NC include white-boards, chat-rooms, telepresence solutions, etc. All these are combined with actual end-user applications software to form NC end-to-end solutions.

5.2.2 Internet and Intranets

The Internet is a WAN. It is based on the IP protocol discussed above. An intranet may be loosely defined as a network that is functionally just like the Internet, but restricted to the computational environment of an enterprise. Intranets thus promise higher security than the Internet by blocking of access from outside. However, they support WWW-like hypertext. Because of their additional security, Intranets are used within enterprises to manage internal processes. For example, they are used to define sets of forms, link them, and execute them appropriately with respect to the enterprise's business models. When the ongoing processes in the enterprise are made explicit, the resulting system is sometimes termed a *process-driven intranet* [87].

Intranets, for most part, tend to be LANs. However, intranets of some bigger companies span counties, states and even continents, and use technology that is more appropriate for WANs.

5.2.3 Network Performance

One of the common ailments of the WAN that today comprise the Internet is that the resources are over-subscribed and that during peak load times this translates into lowered quality of service that end-users see. Figure illustrates the problem. The vertical axis shows the delay in receiving the same WWW page between East and West Coast of the US during different times of the day. The horizontal axis is in 30 minute intervals starting at midnight on Friday (Saturday 0 hours). We see that during weekdays delays can be unacceptably long during peak hours of 8 AM to 8 PM. it is obvious that if one wishes to use the Internet for adequate support of advanced NC applications, such as distance education or collaborative interactive work (where 250 ms may be excessive for a keystroke return), we need an infrastructure that will actively support quality of service for different categories of applications and users. Internet 2, the next generation Internet infrastructure is expected to provide adequate management of the resources that would, for applications that need it, essentially flatten the Figure 4 curves to an acceptable (and nearly constant) response time. This will require cooperation of not only the networking layers but also of the applications layers and applications themselves. Internet 2 is expect to start operating in the near future over OC48 (2.4 Gbps) and OC192 (10 Gbps) backbone meshes, and its edge and core devices will entertain protocols and mechanisms that will assure adequate user-level quality of service.

5.3 Network Functionalities

Effective NC relies on a number of other functionalities being available essentially as standardized services. This is so application developers do not have to implement them, and so they produce repeatable and reliable computations.

One set of functionalities is termed *security* [22, 80]. It includes support for authenticating the senders and receivers of messages and ensuring access control to prevent unauthorized users from reading or modifying data [70]. Other services include those for *digital cash*, and for guaranteeing that a user cannot repudiate their online commitments.

5.4 Standards

There are a number of relevant standards. Some of these are covered in the above; for the others, we defer the discussion to the cited materials.

6 Research Issues

We have taken a workflow-centered stance toward NC. This stance leads naturally to two major classes of research issues, corresponding to the design and analysis of workflows, respectively. We discuss some specific challenges and emerging approaches below. Included is also the issue of how analysis techniques might feed into the design.

6.1 Formal Methods

Because of the importance and complexity of NC applications, it is clear that techniques are required to construct them in a robust manner. Formal methods as studied in software engineering and logics of programs are, therefore, of great potential value. Formal methods can apply in various ways, such as

- capturing requirements precisely
- verifying individual application programs—this is no different than verification in general
- verifying protocols for the interaction of different components of an NC application
- constructing small, but critical subsystems in a robust manner.

Techniques based on variants of temporal logic appear especially attractive for the above purposes [25, 36, 48].

Although formal methods and tools have much potential, their present incarnations can usually handle only fairly small systems. NC applications could have the advantage in this regard, because through the careful application of autonomy the details of the interacting modules may be hidden behind well-defined and compact system interfaces. However, NC applications also require a wider range of abstractions with which to program. These too have an obvious connection with formal methods, and are discussed below.

6.2 Interaction-Oriented Programming

Computing is in the midst of a paradigm shift. After decades of progress on representations and algorithms geared toward individual computations, the emphasis is shifting to what lies among the computations, and between the computations and their environment. This emerging paradigm is that of *interaction*. Interaction goes beyond traditional parallel or distributed computing, because it models the component computations as autonomous and able to interface with an environment [111, 110].

Interaction is precisely the metaphor with which to view computation in settings where network computing reigns. Megaprogramming refers to programming large system at the component level [114]. This idea is essentially a form of software engineering in which large modules are composed to form a large heterogeneous system. This is an inspiring vision, and is addressed by the following discussion.

We suggested above that multiagent systems address many of these challenges quite naturally. But how may we construct multiagent systems efficiently, so that the generate the flexible and felicitously interacting workflows that we desire? If the agents are constructed modularly, the main challenge is in specifying and generating the right interactions. We term our approach *interaction-oriented programming (IOP)*, and include in it abstractions and techniques that capture the structure of the desired interactions and preserve the agents' autonomy. We tentatively identify three layers of IOP, from lower to upper:

- *coordination*, which enables the agents to operate in a shared environment
- *commitment*, which reflects the agents' obligations to one another, capturing the social structure of multiagent systems and the norms governing their behavior
- *collaboration*, which deals with knowledge-level constraints on communications.

Some informal concepts, such as competition, may be classified into different layers: bidding in an auction requires no more than coordination, whereas commerce involves commitments, and negotiation involves sophisticated protocols.

Concepts related to the above have been studied in areas as diverse as distributed computing (DC), databases (DB), and distributed artificial intelligence (DAI). The DB and DC work focuses on narrower problems of synchronization, and eschews high-level concepts such as social commitments. Thus it is less flexible, but more robust, than the DAI work. However, the complexity of NC applications demands flexibility and their importance demands rigor. IOP is about programming abstractions that achieve both of these properties.

The coordination layer of IOP deals with specifying *skeletons* as high-level descriptions of agent behavior that are relevant to the agent's interactions with one another. The skeletons define what events of an agent are visible externally, and what properties they have. The agents interact with each other, but programatically the interactions appear to be through a distributed coordination service.

The properties specify what the other agents can expect: roughly, whether they can request a task corresponding to an event, ask the agent not to perform a task, or ask the agent to put off doing something till they are ready. Examples of events are whether the agent will start a task (which others can request) or declare a failure (which the other cannot prevent) or declare success (which the others can ask this agent not to do until they are ready). Based on the events, coordination specification scan be stated that determine the acceptable behaviors of the multiagent system. The key research challenges are in defining expressive such efficiently processable languages for specifying coordination requirements, producing efficient algorithms for them, and relating the corresponding models to methodologies for system development [93, 94].

The commitment layer introduces the logical notion of a "social" commitment, which is a relationship between two agents. One or both of the agents could be a multiagent system. Agents enter into commitments voluntarily either by saying so, or by adopting predefined roles that require those commitments. Commitments can be canceled or updated provided any additional metacommitments are satisfied. The key research challenges are in creating formalisms in which a useful range of commitments can be expressed and efficiently processed, and in corresponding methodologies for system development [92, 95].

6.3 Quality of Service

Traditional, i.e., network-related, QoS is defined using measures such as keystroke delays, probability of loss of data, jitter, and throughput. Because of the increasing focus on end-user workflows in NC, we broaden the classical definition of QoS to include end-user quality characteristics such as system reliability and availability, performance, algorithmic scalability, effectiveness, and quality of user-system interactions. For example, users find keystroke round-trip delays of more than 250 ms unacceptable [11, 55, 33]. In education, for lessons of about an hour, studies indicate that of good user satisfaction requires a probability of error < 0.05, whereas even a probability > 0.14 is unacceptable.

To make NC applications appear truly "appliance-like," the systems must support negotiation of extended QoS guarantees for users, monitor QoS, and adjust resource allocation dynamically [32]. This is a complex challenge, because of the nature of applications and the fact that the infrastructureis distributed and heterogeneous. Existing techniques for application layer QoS management are limited to static allocation of resources. These are not suitable for the above challenge. Thus, in addition to reactive mechanisms for control of QoS violations, predictive QoS management is required [69].

For instance, although ATM solutions are intended to provide QoS support, only applications written specially for ATM can take full advantage of its QoS capabilities. Some approaches seek to establish end-to-end connections over existing IP solutions (e.g., RSVP, AREQUIPA) [1, 116].

Ongoing discussions on QoS provisioning have often debated whether the network can be dimensiond such that traffic requirements are unlikely to ever exceed the available resources. However, the "infinite resources" argument has never been found to be true.

Dixit argues that there is a considerable need to monitor and model QoS not only at the network layer, but at the application layer as well [32]. While guaranteed bounds are often achievable and provisioned for in data networks (e.g., [72]), they are usually intended for network-to-network connections (see Figure 3) and do not account for QoS violations resulting form end-user applications (including server and client user-level effects). Further, provisioning for such bounds is expensive. Tolerant and adaptive clients will be unwilling to pay excessive tariffs required to suport these bounds. In addition, reserving bandwidth in advance may lead to wasted resources. Another significant issue is that the overal stochasticity of the NC systems is usually not accounted for in most existing approaches. The stochasticity which results in distributed systems due to scheduling mechanisms, resource dynamics, and so on, may manifest as poor QoS and may be tagged as QoS failures unless it is explicitly incorporated in QoS models. Application layer monitoring is needed to provide reliable predictive estimates of QoS in the future.

Therefore, in emerging networks, QoS should be provisioned dynamically and in a model using more unified approaches, which account for the combined effects of interacting NC components and layers.

6.4 Performance of NC Applications

The performance on an NC application can be defined in user terms (e.g., availability or reliability [32]) or more basic parameters (e.g., latency and throughput). There are many factors that influence performance of NC. These include the performance of (1) the networking infrastructure and host hardware and operating system [104, 101], (2) the transport and networking software [104, 101], and (3) the middleware, which resides between the transport layer and the application [5]. The impact of network congestion on response delay is shown in Figure fig-internet-congestion. However, one can elicit an equally poor performance simply through inadequate tuning or weak workstations [83]. It is the combination of the above factors and the operational profile of the fielded application that determines its performance.

The overhead imposed by the middleware is related to its functionality. *Lightweight* middleware provides only a subset of functions, e.g., for inter-process communication, necessary for the support of the immediate application needs. *Full-function* middleware makes all operations and communication modes available, even if not needed by an application. This typically leads to greater communication overhead than in specialized lightweight layers. Thus, functionally equivalent middleware implementations may have vastly differing performance profiles.

The following results compare the delay in sending a null-body asynchronous (one-way) message, and in sending a null-body request-reply (two-way) message. Lightweight Software Bus (LSB) is a communication layer developed at NCSU [5, 6]. SOMObjects, Orbix, and ORBeline are commercial CORBA-compliant products. SII and DII are the static and dynamic invocation interfaces defined in CORBA.

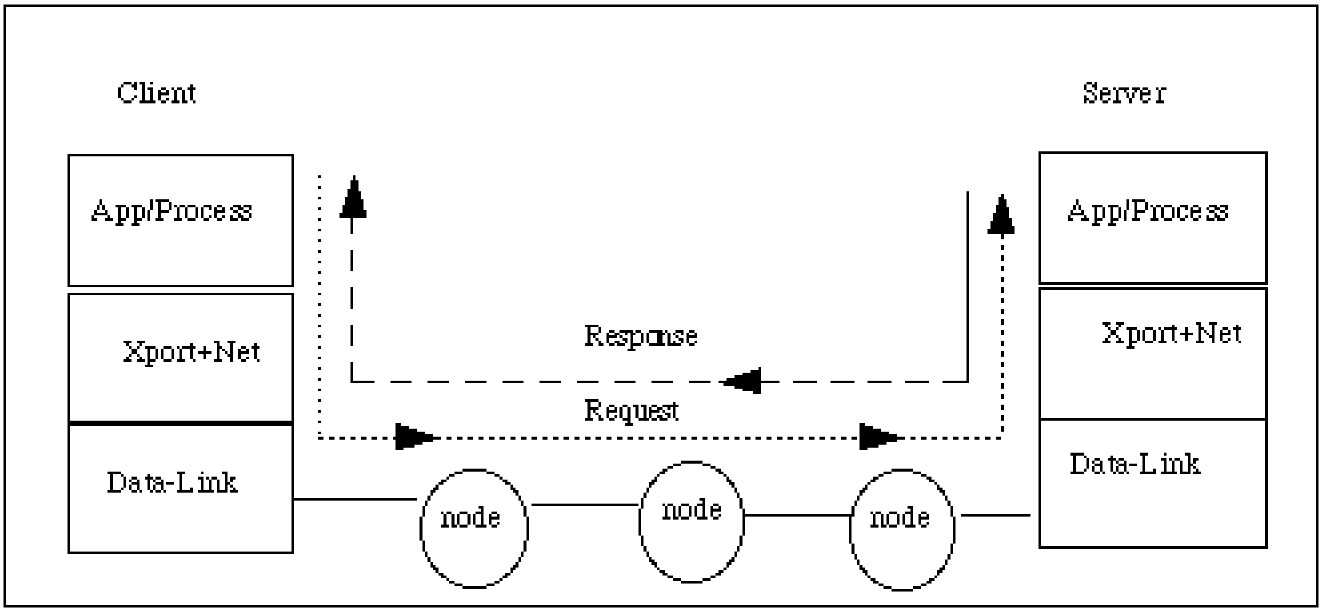

Figure 3: An illustration of the path (and overhead) between an NC client and another top-layer component (e.g., server)

	LSB	SOM SII	SOM DII
One-way Latency	0.20 ms	2.04 ms	2.06 ms
Two-way Latency	1.83 ms	4.32 ms	5.26 ms

Table 2: Latency Measurements for LSB and SOMObjects

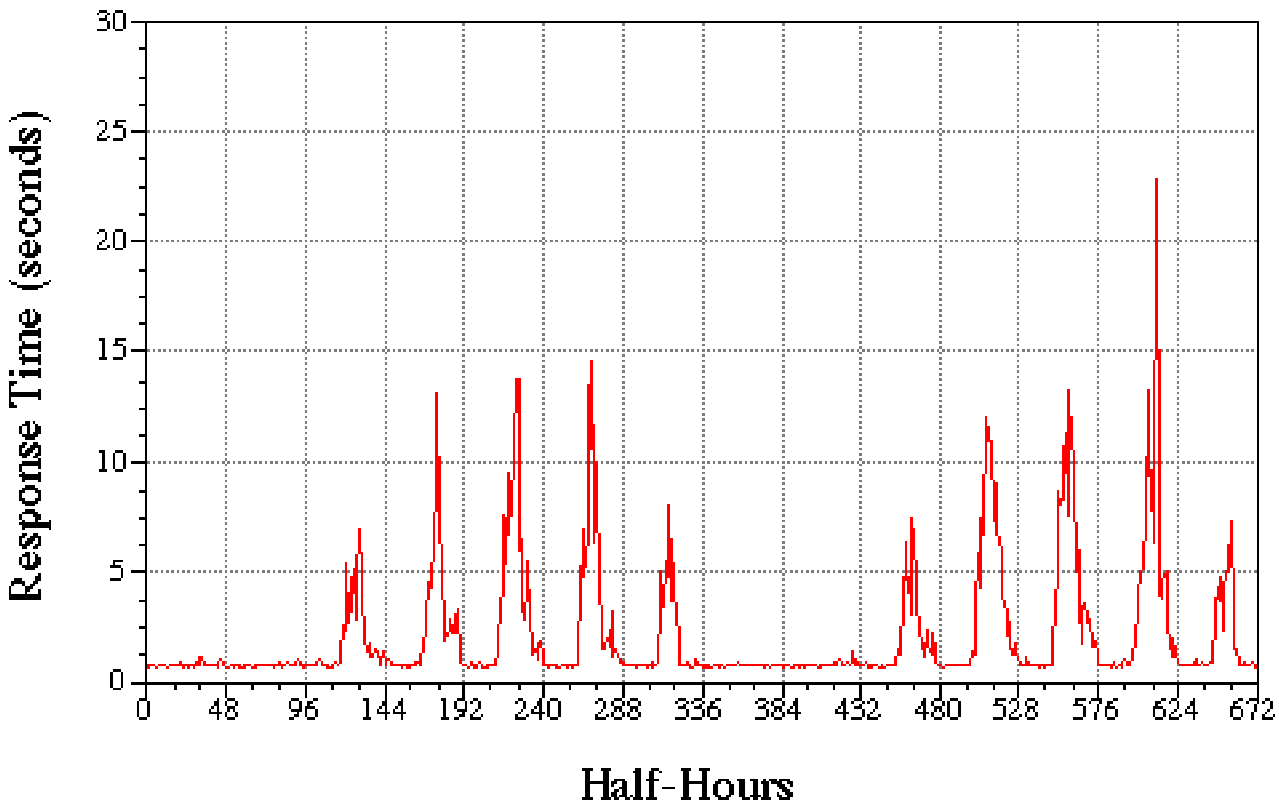

Figure 4: Illustration of Internet Congestion Delays

	Orbix SII	Orbix DII	ORBeline SII	ORBeline DII
One-way Latency	0.5 ms	0.4 ms	0.1 ms	0.1 ms
Two-way Latency	2.7 ms	10.5 ms	2.0 ms	2.0 ms

Table 3: Latency Measurements for Orbix and ORBeline

Table 2 shows that the delay incurred by a message traversing LSB is much less than for SOMObjects [5, 6]. Also, for a two-way operation, DII yields a significantly higher latency than SII. Table 3 shows similar comparisons on two other CORBA products. These results are extracted from the graphs in [43] for parameter-less method invocation. The tables show the ranges of overheads that are possible.

7 Reading List

Network computing is a huge area that impinges upon a number of subfields of computing. We include a list of reading to obtain additional information about the various topics discussed above.

- Multidatabases: [57, 75, 17, 97, 115]
- Traditional and extended transactions: [46, 35, 57]
- Workflow modeling and enactment: [26, 40, 44, 2, 67, 86]
- Agents and multiagent systems: [85, 53, 51, 112, 34, 23]
- Distributed objects and CORBA: [57, 71, 90]
- Applications of network computing: [21, 27, 56, 79]
- Scientific computing: [38, 66, 100, 39]
- Software and reliability engineering: [47, 12, 62, 63]
- Performance evaluation: [106, 6, 103]
- Networking and quality of service: [101, 74]
- Architectures: [102, 8, 56, 87]

8 Cross References

These are terms that should refer to this article.

1. Internet computing
2. Open information environments
3. Problem-solving environments
4. Workflow management
5. Agents
6. Multiagent systems
7. Middleware
8. Quality of service
9. Internet and Internet 2 applications

References

[1] W. Almesberger. AREQUIPA: Design and implementation. TR 96/213, EPFL, ftp.lrcwww.epfl.ch/ pub/ arequipa, November 1996.

[2] Paul C. Attie, Munindar P. Singh, Amit P. Sheth, and Marek Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proceedings of the 19th VLDB Conference*, pages 134–145, August 1993.

[3] John L. Austin. *How to Do Things with Words*. Clarendon Press, Oxford, 1962.

[4] Mark Baker and Geoffrey Fox. Metacomputing: The informal supercomputer. In *Proceedings of the NSF Train the Trainer Workshop*, Ithaca, NY, May 1996. Cornell Theory Center. http:// renoir.csc.ncsu.edu/ RTCPP/ HTML/ Workshop2/ index.html.

[5] Rajini Balay. *A Lightweight Middleware Architecture and Evaluation of Middleware Performance*. PhD thesis, North Carolina State University, Raleigh, 1998.

[6] Rajini Balay, Mladen A. Vouk, and Harry Perros. Implications of middleware and application level communication issues on the performance of network-based problem-solving environments. In Elias Houstis and John R. Rice, editors, *Problem Solving Environments*, Computational Science and Engineering. IEEE Press, Los Alamitos, CA, 1998.

[7] Carlo Batini, Stefano Ceri, and Shamkant Navathe. *Conceptual Database Design*. Benjamin Cummings, Redwood City, CA, 1992.

[8] Miro Benda, editor. *Special Issue on Internet Architecture*, volume 2(2) of *IEEE Internet Computing*. March 1998. Contains 6 articles.

[9] T. Berners-Lee, R. Cailliau, J. F. Groff, and B. Pollerman. World-wide web: The information universe. *Electronic Networking: Research, Applications and Policy*, 1(2):52–58, Spring 1992.

[10] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, August 1994.

[11] M. D. Bitzer and D. L. Bitzer. Teaching nursing by computer: An evaluation study. *Computers in Biology and Medicine*, 3(3):187–204, 1973.

[12] Barry W. Boehm. *Software Risk Management*. IEEE Computer Society Press, Los Alamitos, 1989.

[13] C. Mic Bowman, Peter B. Danzig, Udi Manber, and Michael F. Schwartz. Scalable internet resource discovery: Research problems and approaches. *Communications of the ACM*, 37(8):98–114, August 1994.

[14] S. Bradner and A. Mankin. Recommendation for the IP next generation protocol, January 1995. RFC1752.

[15] Alejandro Buchmann, M. Tamer Özsu, Mark Hornick, Dimitrios Georgakopoulos, and Frank A. Manola. A transaction model for active distributed object systems. In *[35]*, chapter 5, pages 123–158. 1992.

[16] Omran A. Bukhres, Jiansan Chen, Weimin Du, Ahmed K. Elmagarmid, and Robert Pezzoli. InterBase: An execution environment for heterogeneous software systems. *IEEE Computer*, 26(8):57–69, August 1993.

[17] Omran A. Bukhres and Ahmed K. Elmagarmid, editors. *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*. Prentice-Hall, 1996.

[18] Christoph Bußler and Stefan Jablonski. An approach to integrated workflow modeling and organization modeling in an enterprise. In *Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. IEEE Computer Society Press, 1994.

[19] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[20] Nicholas Carriero and David Gelernter. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[21] Brahim Chaib-draa. Industrial applications of distributed artificial intelligence. In *[51]*, pages 31–35. 1997. (Reprinted from *Communications of the ACM, 1995*).

[22] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985. Extended version with subtitle "Card Computers to Make Big Brother Obsolete" available at `http://digicash.support.nl/publish/bigbro.html`.

[23] David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant agents for mobile computing. In *[51]*, pages 267–282. 1997. (Reprinted from *IEEE Personal Communications, 1995*).

[24] Panos Chrysanthis and Krithi Ramamritham. ACTA: The SAGA continues. In *[35]*, chapter 10, pages 349–397. 1992.

[25] E. Clarke, O. Grumberg, and D. Long. Model checking. In *Proceedings of the International Summer School on Deductive Program Design*, pages 428–439, 1990.

[26] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[27] Mark R. Cutkosky, Robert S. Englemore, Richard E. Fikes, Michael R. Genesereth, Thomas R. Gruber, William S. Mark, Jay M. Tenenbaum, and Jay C. Weber. PACT: An experiment in integrating concurrent engineering systems. In *[51]*, pages 46–55. 1997. (Reprinted from *IEEE Computer, 1993*).

[28] Peter B. Danzig, Shih-Hao Li, and Katia Obrazacka. Distributed indexing of autonomous internet services. *Computing Systems*, 5(4):433–459, Fall 1992.

[29] Umesh Dayal, Hector Garcia-Molina, Mei Hsu, Ben Kao, and Ming-Chien Shan. Third generation TP monitors: A database challenge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1993. Industrial track paper.

[30] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification, December 1995. RFC1883.

[31] R.L. Dennis, D. W. Byun, J. H. Novak, K. J. Galluppi, C. C. Coats, and M. A. Vouk. The next generation of integrated air quality modeling: EPA's Models-3. *Atmospheric Environment*, 30(12):1925–1938, 1996.

[32] Paritosh Dixit. *Quality of Service Modeling for Wide Area Network Based Systems*. PhD thesis, North Carolina State University, Raleigh, 1998.

[33] Paritosh Dixit, Mladen A. Vouk, and Donald L. Bitzer. Reliability behavior of a large network-based education system. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 43–56. IEEE Computer Society Press, November 1997.

[34] Edmund H. Durfee, Daniel L. Kiskis, and William P. Birmingham. The agent architecture of the University of Michigan digital library. In *[51]*, pages 98–108. 1997. (Reprinted from *Proceedings of the IEE: Software Engineering, 1997*).

[35] Ahmed K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, San Mateo, 1992.

[36] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. North-Holland, Amsterdam, 1990.

[37] Foundation for intelligent physical agents (FIPA) 1997 specification, 1998. http://www.fipa.org.

[38] Efstratios Gallopoulos, Elias Houstis, and John R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 1:11–23, Summer 1994.

[39] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, 1994.

[40] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–152, April 1995.

[41] Dimitrios Georgakopoulos, Mark F. Hornick, and Frank Manola. Customizing transactions models and mechanisms in a programmable environment supporting reliable workflow automation. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):630–649, August 1996.

[42] Charles R. Gilman, Manuel Aparicio, J. Barry, Timothy Durniak, Herman Lam, and Rajiv Ramnath. Integration of design and manufacturing in a virtual enterprise using enterprise rules, intelligent agents, STEP, and work flow. In *SPIE Proceedings on Architectures, Networks, and Intelligent Systems for Manufacturing Integration*, pages 160–171, 1997.

[43] Aniruddha Gokhale and Douglas C. Schmidt. Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks. In *17th International Conference on Distributed Systems*, May 27-30 1997.

[44] Esin Gokkoca, Mehmet Altinel, Ibrahim Cingil, E. Nesime Tatbul, Pinar Koksal, and Asuman Dogac. Design and implementation of a distributed workflow enactment service. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, pages 89–98, 1997.

[45] Art Goldschmidt, Martin Hardwick, Jon Judd, Herman Lam, Jeff Pan, Jay Ramanathan, Tom Rando, Stanley Su, Craig Thompson, and David Zenie. NIIIP reference architecture: Concepts and guidelines. TR NTR95-01, National Industrial Information Infrastructure Protocols (NIIIP) Consortium, January 1995. www.niiip.org/ public-forum/ NTR95-01/.

[46] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, 1993.

[47] Maurice H. Halstead. *Elements of Software Science*. Elsevier, Amsterdam, 1977.

[48] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–296, May 1997.

[49] Meichun Hsu, editor. *Special Issue on Workflow and Extended Transaction Systems*, volume 16(2) of *Bulletin of the IEEE Technical Committee on Data Engineering*. June 1993. Contains 13 articles.

[50] Michael N. Huhns and Munindar P. Singh. Ontologies for agents. *IEEE Internet Computing*, 1(6):81–83, December 1997. Instance of the column *Agents on the Web*.

[51] Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, 1997.

[52] Akhtar Jameel, Matthias Stuempfle, Daniel Jiang, and Axel Fuchs. Web on wheels: Toward internet-enabled cars. *IEEE Computer*, 31(1):69–76, January 1998.

[53] Nicholas R. Jennings and Michael J. Wooldridge, editors. *Agent Technology: Foundations, Applications, Markets*. Springer-Verlag, Berlin, 1997.

[54] Mohan Kamath and Krithi Ramamritham. Bridging the gap between transaction management and workflow management. In *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions*, May 1996. `http:// optimus.cs.uga.edu:5080/ activities/NSF-workflow/ kamath.html`.

[55] P. K. Kauer. An analysis of north carolina state university's network performance. Master's thesis, Department of Computer Science, North Carolina State University, Raleigh, 1995.

[56] Ken Kennedy, Charles F. Bender, John W. D. Connolly, John L. Hennessy, Mary K. Vernon, and Larry Smarr. A nationwide parallel computing environment. *Communications of the ACM*, 40(11):63–72, November 1997.

[57] Won Kim, editor. *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press and Addison-Wesley, New York, 1994. Reprinted with corrections, 1995.

[58] Yannis Labrou and Tim Finin. Semantics and conversations for an agent communication language. In *[51]*, pages 235–242. 1997. (Reprinted from *Proceedings of the International Joint Conference on Artificial Intelligence, 1997*).

[59] Ted G. Lewis. Information appliances: Gadget netopia. *IEEE Computer*, 31(1):59–68, January 1998.

[60] Frank Leymann and Wolfgang Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.

[61] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. ACM Press and Addison-Wesley, New York, 1996.

[62] Michael R. Lyu, editor. *Software Fault-Tolerance*. John Wiley, New York, 1995.

[63] Michael R. Lyu, editor. *Handbook of Software Reliability Engineering*. McGraw-Hill and IEEE Computer Society, New York, 1996.

[64] Steve Mann. Wearable computing: A first step toward personal imaging. *IEEE Computer*, 30(2):25–32, February 1997.

[65] Steven McCanne and Van Jacobson. VIC: A flexible framework for packet video. In *Proceedings of the ACM Multimedia Conference*, pages 511–522, 1995.

[66] Gregory J. McRae. How application domains define requirements for the grid. *Communications of the ACM*, 40(11):75–83, November 1997.

[67] Raúl Medina-Mora and Kelly W. Cartron. ActionWorkflow^R in use: Clark County department of business license. In *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, pages 288–294, February 1996.

[68] J. D. Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 10(2):14–32, March 1993.

[69] K. Nahrstedt. *An Architecture for End-to-End Quality of Service and its Experimental Validation*. PhD thesis, University of Pennsylvania, Philadelphia, 1995.

[70] B. C. Neumann and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9), September 1994.

[71] M. Tamer Özsu, Umeshwar Dayal, and Patrick Valduriez, editors. *Distributed Object Management*. Morgan Kaufmann, San Mateo, 1994.

[72] A. K. Parikh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services network. *IEEE Transactions on Networking*, 2(2):137–150, April 1994.

[73] Ramesh S. Patil, Richard E. Fikes, Peter F. Patel-Schneider, Don McKay, Tim Finin, Thomas Gruber, and Robert Neches. The DARPA knowledge sharing effort: Progress report. In *[51]*, pages 243–254. 1997. (Reprinted from *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, 1992*).

[74] Charles E. Perkins. Mobile networking through mobile IP. *IEEE Internet Computing*, 2(1):58–69, January 1998.

[75] Evaggelia Pitoura, Omran A. Bukhres, and Ahmed K. Elmagarmid. Object-oriented multidatabase systems: An overview. In *[17]*, chapter 10. 1996.

[76] J. Postel. Internet protocol, September 1981. RFC0791.

[77] Parallel virtual machine. http://www.netlib.org/pvm3, 1997.

[78] Krithi Ramamritham and Panos K. Chrysanthis. A taxonomy of correctness criteria in database applications. *The VLDB Journal*, 5(1):85–97, January 1996.

[79] Daniel A. Reed, Roscoe C. Giles, and Charles E. Catlett. Distributed data and immersive collaboration. *Communications of the ACM*, 40(11):39–48, November 1997.

[80] Michael K. Reiter. Distributing trust with the Rampart toolkit. In *[51]*, pages 306–309. 1997. (Reprinted from *Communications of the ACM, 1996*).

[81] John R. Rice and Ronald F. Boisvert. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering*, 3:44–53, Fall 1996.

[82] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, January 1998.

[83] A. J. Rindos, S. Woolet, D. Cosby, L. Hango, and M. Vouk. Factors influencing ATM adapter throughput. *Multimedia Tools and Applications*, 2(3):253–272, 1996.

[84] Nancy Ross-Flannigan. The virtues (and vices) of virtual colleagues. *MIT Technology Review*, March 1998.

[85] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, NJ, 1995.

[86] Sunil K. Sarin. Workflow and data management in InConcert. In *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, pages 497–499, February 1996.

[87] Walt Scacchi and John Noll. Process-driven intranets: Life-cycle support for process reengineering. *IEEE Internet Computing*, 1(5):42–49, October 1997.

[88] Michael F. Schwartz, Alan Emtage, Brewster Kahle, and B. Clifford Newman. A comparison of internet resource discovery approaches. *Computing Systems*, 5(4):461–493, Fall 1992.

[89] Hyong Sop Shim, Robert W. Hall, Atul Prakash, and Farnam Jahanian. Providing flexible services for managing shared state in collaborative systems. In *Proceedings of the European Conference on Computer Supported Cooperative Work (ECSCW)*, Lancaster, UK, 1997.

[90] Jon Siegel. *CORBA: Fundamentals and Programming*. Object Management Group and Wiley, New York, 1996.

[91] Munindar P. Singh. Synthesizing distributed constrained events from transactional workflow specifications. In *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, pages 616–623, February 1996.

[92] Munindar P. Singh. Commitments among autonomous agents in information-rich environments. In *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, pages 141–155. Springer-Verlag, May 1997.

[93] Munindar P. Singh. A customizable coordination service for autonomous agents. In *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL)*. Springer-Verlag, July 1997. Publication date: 1998.

[94] Munindar P. Singh. Developing formal specifications to coordinate heterogeneous autonomous agents. In *Proceedings of the 3rd International Conference on Multiagent Systems (ICMAS)*. IEEE Computer Society Press, July 1998.

[95] Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 1998. In press.

[96] Munindar P. Singh. Principles of agent communication. *IEEE Computer*, 1998. To appear.

[97] Munindar P. Singh, Philip E. Cannata, Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Kayliang Ong, Amit P. Sheth, Christine Tomlinson, and Darrell Woelk. The Carnot heterogeneous database project: Implemented applications. *Distributed and Parallel Databases: An International Journal*, 5(2):207–225, April 1997.

[98] Munindar P. Singh and Michael N. Huhns. Automating workflows for service provisioning: Integrating AI and database technologies. *IEEE Expert*, 9(5):19–23, October 1994.

[99] Smart description. http://smart.npo.org/, 1997.

[100] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, 1995.

[101] William Stallings. *Data and Computer Communications*. Prentice-Hall, Upper Saddle River, NJ, 5th edition, 1997.

[102] Rick Stevens, Paul Woodward, Tom DeFanti, and Charlie Catlett. From the I-way to the national technology grid. *Communications of the ACM*, 40(11):51–60, November 1997.

[103] W. Richard Stevens. *Queueing Networks with Blocking: Exact and Approximate Solutions*. Oxford University Press, New York, 1994.

[104] W. Richard Stevens. *TCP/IP Illustrated, Volumes 1–3*. Addison-Wesley, Reading, MA, 1994.

[105] Stanley Y. W. Su, Herman Lam, Tsae-Feng Yu, Javier A. Arroyo-Figueroa, Zhidong Yang, and Sooha Lee. NCL: A common language for achieving rule-based interoperability among heterogeneous systems. *Journal of Intelligent Information Systems (JIIS)*, 6(2/3):171–198, 1996.

[106] Y. Viniotis and H. Perros, editors. *Special Issue on ATM networks and Their Performance*. Performance Evaluation Journal. 1998.

[107] M. Vouk, J. Hines, A. J. Rindos, S. Woolet, and J. Lester. ATM technology enabling educational applications across the North Carolina information highway. In *Proceedings of the 7th World Telecommunications Forum*, pages 519–522, Geneva, October 1995. International Telecommunications Union.

[108] Jacques Wainer, Mathias Weske, Gottfried Vossen, and Claudia M Bauzer Medeiros. Scientific workflow systems. In *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions*, May 1996. `http:// optimus.cs.uga.edu:5080/ activities/NSF-workflow/ wainer.html`.

[109] Web lecture system. http://renoir.csc.ncsu.edu/WLS, 1997.

[110] Peter Wegner. Interactive foundations of computing. *Theoretical Computer Science*, May 1997. In press. Preprint at www.cs.brown.edu/ people/ pw/.

[111] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, May 1997.

[112] Gerhard Weiß, editor. *Introduction to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1998.

[113] Gio Wiederhold. Mediators in the architecture of future information systems. In *[51]*, pages 185–196. 1997. (Reprinted from *IEEE Computer, 1992*).

[114] Gio Wiederhold, Peter Wegner, and Stefano Ceri. Toward megaprogramming. *Communications of the ACM*, 35(11):89–99, November 1992.

[115] Darrell Woelk, Philip Cannata, Michael Huhns, Nigel Jacobs, Tomasz Ksiezyk, Greg Lavender, Greg Meredith, Kayliang Ong, Wei-Min Shen, Munindar Singh, and Christine Tomlinson. Carnot prototype. In *[17]*, chapter 18, pages 621–648. 1996.

[116] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 9(5), September 1993.

[117] Computer Select Ziff. Computer select – software product specification: Data sources report, September 1994. Ziff Communications Company, September 1994.