A distributed system architecture for a distributed application environment

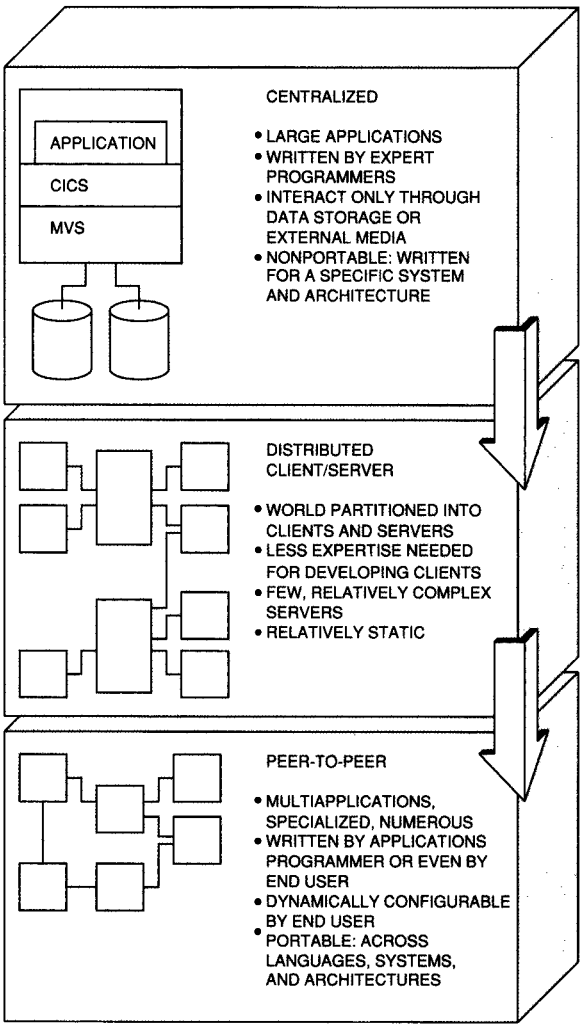
by M. A. Bauer
N. Coburn
D. L. Erickson
P. J. Finnigan
J. W. Hong
P.-Å. Larson
J. Pachl
J. Slonim
D. J. Taylor
T. J. Teorey

Advances in communications technology, development of powerful desktop workstations, and increased user demands for sophisticated applications are rapidly changing computing from a traditional centralized model to a distributed one. The tools and services for supporting the design, development, deployment, and management of applications in such an environment must change as well. This paper is concerned with the architecture and framework of services required to support distributed applications through this evolution to new environments. In particular, the paper outlines our rationale for a peer-to-peer view of distributed systems, presents motivation for our research directions, describes an architecture, and reports on some preliminary experiences with a prototype system.

Continuous advances in communications technology coupled with the development of powerful desktop workstations are fueling the growth of distributed computing. Users' demands for transparent access to information and applications, regardless of the hosts on which they reside, require interoperability among heterogeneous hosts, operating systems, and data sources. The development of distributed applications in such environments presents many challenges to the developers of applications and to the providers of computing and development environments.

Developers of distributed applications must often cope with details of protocols, differing data representations, multiple communication standards, and more. Development tools (such as languages, test case generators, and debuggers) are often limited in their support for developing distributed applications. Even when distributed applications are made to work, their ongoing management and operation become challenges requiring sophisticated expertise to overcome performance problems, changing systems, etc.

Figure 1 Evolution of the computing environment



Providers of distributed computing and development environments, therefore, must supply services that hide the myriad of details from the application developers and that enable the development of distributed applications. There are, however, many questions about the nature of future distributed applications and their supporting services: What are the services required by distributed applications? How should these services be designed and implemented to accommodate openness, scalability, and manageability? How should services be distributed? How are different sets of services related? What is a suitable architecture for multiple services that can accommodate both existing systems and the emergence of new technology?

The CORDS research project is an effort to understand the problems and challenges that are central to the development of environments for the design, development, and management of distributed applications. It brings together researchers from four IBM research laboratories, six Canadian universities, four American universities, and other international research centres. The acronym "CORDS" stems from the original name for the group: "COnsortium for Research on Distributed Systems." Even though the official name of the project, i.e., the name on the original proposal, was "Alliance for Research in Distributed Systems," the acronym for the group was kept.

The scope of the research encompasses both new techniques for developing distributed applications and for understanding the services required by distributed applications and the associated tools. Included in the latter category are the integration and distribution requirements of both applications and support tools. Other research efforts have addressed problems in these areas as well; for example, Advanced Networked Systems Architecture (ANSA**),[1,2] Distributed Computing Architecture,[3] Common Applications Environment (CAE),[4] and Open Software Foundation Distributed Computing Environment (OSF DCE**).[5]

CORDS is unique in two fundamental aspects. First, CORDS takes a basic premise that distributed environments and applications using these environments are evolving from an environment with *client/server* interactions to one with *peer-to-peer* interactions. Second, it brings together researchers with different expertise in distributed computing in order to understand the trade-offs, boundaries, and interplay between different sets of services and applications. The researchers have expertise in a number of areas: (distributed) databases, programming languages, (distributed) systems, and visualization techniques.

This paper is concerned with the architecture and framework of services required in a distributed application development environment. It outlines the rationale for our peer-to-peer view of distributed systems, presents motivation for the research directions, and describes the architecture. The architecture emerging from the research to

date serves as a blueprint to guide researchers in developing prototypes and investigating important integration problems. Its definition is continuously being refined to reflect the ongoing research within CORDS addressing more specific questions involving, for example, multidatabases,[6,7] application management,[8–11] distributed debugging,[12,13] and visualization.[14–16]

The paper is organized as follows. First we discuss the motivation for the CORDS project and describe our long-term view of distributed application development and management environments. Following that is a brief overview of other efforts in defining distributed architectures and associated services. Then the CORDS architecture is introduced. Afterward a brief overview is given of a "proof of concept" prototype developed within the CORDS project, relating its services to components with the CORDS architecture.

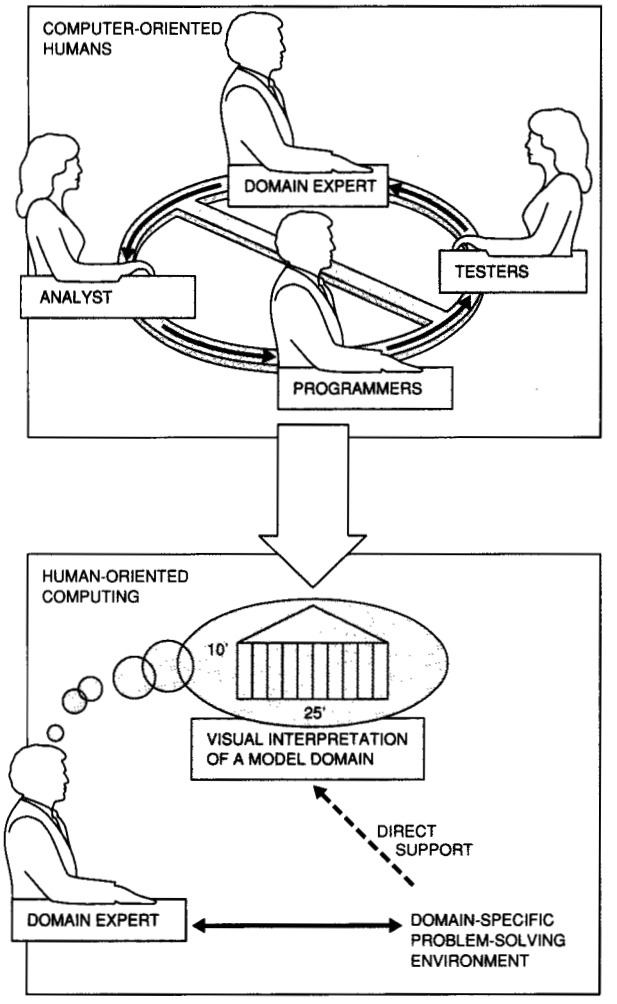
Motivation: shifting paradigms

Distributed systems with tens or even hundreds of thousands of computing nodes interacting over vast communication channels are already possible. As technology continues to advance, more and more powerful computers will be available to individuals. In turn, these computers will have access via high-speed communications to a vast array of computing resources.

This view of the computing environment is not radical but is rather a natural evolution of the computing field itself (see Figure 1). Initially, computing was done on large, centralized systems with limited access. This phase gave way to time-sharing environments and in turn to networked environments capable of supporting client/server applications. In each phase of this evolution, computing power has been brought closer to the individual user. Intertwined with this evolution of computing technology is the manner in which users interact with the computing environment and the applications available to the user. More powerful computers, advances in interface technology, and more sophisticated applications have meant that the number of users capable of using computing and information resources is constantly increasing.

Just as the computing environment has evolved, we see human-computer interaction evolving from one that required *computer-oriented* humans, to one best described as *human-oriented* computing (see Figure 2). Such a fundamental paradigm shift in the way computing is viewed has two significant implications. First, the end user will become increasingly more important in developing applications. Second, *domain specialists* will emerge and will be responsible for developing domain-specific toolkits. These toolkits will consist of domain-specific building blocks. As end users become familiar with these building blocks, they will be able to easily combine them to obtain the desired application functionality. Computing models will be required to permit users to combine components to form customized applications. More importantly for our

Figure 2 Evolution of human-computer interaction



research, underlying computation models, services, and tools must support these specialists and application composition.

We believe that in the long term the partitioning of components into clients and servers will become constraining and that a computing environment based on peer-to-peer interactions will be required. This is not to say that components will not take on *roles* of clients and servers, but rather that the role may change with time and may differ among components; our rationale is further elaborated in the following subsection. Moreover, the shift in the way applications are developed, namely an increased reliance on domain specialists and the composition and customization of applications by end users, will exacerbate the problems in defining, specifying, and integrating the services of future distributed computing systems. Some of the requirements of these future distributed computing systems are discussed in the succeeding subsection.

We see the current environments with client/server interactions as evolving naturally to peer-to-peer interactions.

Evolution to a peer-to-peer environment. Our view of the evolution of the computing environment coupled with the evolution toward a human-oriented computing environment suggests a distributed environment that supports greater human-oriented, end-user computing. The computational model, therefore, should be one that is familiar to end users; the peer-to-peer model satisfies this requirement. Our premise is that the composition of applications will be more natural for end users if the model is peer-to-peer in that this model is natural for many human interactions. It is a more general model of interaction than client/server, since no (artificial) hierarchy of servers and clients needs to be defined.

Moreover, we see the current environments with client/server interactions as evolving naturally to peer-to-peer interactions. In peer-to-peer computing, relationships may be many-to-many. Layering and client/server relationships are not required. Furthermore, each process or component is independent. In some circumstances involving distributed computing, it is easy to see client/server relationships. In many others, however, entities may take on simultaneous roles of both clients and servers. Consider the following example which arose in the context of the CORDS project. The management of large distributed environments requires the collection of data from devices, hosts, etc. Some or all of this information might be stored for subsequent analysis or for historical use, e.g., in modeling. The storage and management of these data can be handled by available database systems. That is, the management components use the services of the database system. Conversely, the database system requires the collection of network and processor performance information in order to effectively optimize the distributed queries; i.e., the database system makes use of the management services. Which is the client and which is the server? An artificial separation of roles leads to either a duplication of services or to a clumsy architecture. From the perspective of the end user or application designer, both components of the system offer services that can be used naturally by the other.

We also feel that a peer-to-peer model can more easily accommodate existing applications. Migrating a centralized application to a client/server model requires, at a minimum, that it be turned into a server that reacts to requests from user clients. This introduces complexities arising with multiple clients and further conversion to accommodate multithreading. In a peer-to-peer environment, the application could be essentially "wrapped" as an entity capable of (perhaps limited) interaction with other applications. The "wrapper" could handle the peer-to-peer communication with other components, leaving the existing application changed little or not at all.

The evolutionary trends in computing and modes of human-computer interaction have led us to consider distributed application development and operational environments based on peer-to-peer interaction. In particular, we have considered the services required of the underlying distributed

system. The nature of these services, the underlying problems in realizing them, the accommodation of existing and emerging applications, and the heterogeneity of computing platforms have provided the motivation for our work. The architecture emerging from this work, though not complete, represents a blueprint for models and prototypes. By studying these models and gaining experience in prototype development and management, we hope to answer some of the fundamental questions surrounding the nature of services required for distributed applications based on peer-to-peer interactions.

Distributed environments must be able to accommodate emerging technologies.

Requirements of future distributed systems. Any environment supporting the development, deployment, and management of distributed applications will have to provide services and be based on an architecture that addresses a number of key requirements of distributed computing systems. An environment based on a human-oriented, peer-to-peer computing model will be no different, although we feel that it will provide a more successful approach in the long term. Following are a set of broad requirements that we feel are most important. These requirements have shaped our research even if we have not yet begun to address some of them directly.

Issues of performance and cost will continue to be important, but other issues, such as reusability and transparency, will be as important. Many of the requirements discussed below are not orthogonal. Compromises and trade-offs among different sets of services will be necessary. Much work remains to understand fully the nature of these compromises and trade-offs as well as to understand how the underlying services are integrated and distributed.

Support peer-to-peer development. The services provided by the distributed environment will have to support the development and operation of distributed applications as collections of peer components. Development tools and languages will be required to support composition of components to form complex applications customized for particular users. The applications and tools will require services to locate components on remote hosts, dynamically connect to and terminate peer-to-peer connections, and migrate components, etc., across heterogeneous computing platforms. The underlying services must provide a simple interface to application developers and domain specialists to reduce the complexity of developing such applications.

Accommodation of legacy applications. The migration of legacy applications to a fully distributed environment will require that existing applications and services evolve rather than be rewritten. The interoperability of existing (centralized) applications and services with distributed applications and services must be addressed. Our view is that within a peer-to-peer environment, such applications and services can be encapsulated, at least logically as a peer process, and integrated with new emerging distributed applications and services.

Accommodation of emerging applications. Conversely, distributed environments must be able to accommodate emerging technologies such as high-speed networks. System administrators should not be constrained to use the "lowest-common-denominator" of technologies and services in their computing environment. In particular, the distributed environment should allow systems developers and administrators to exploit the features of emerging computing environments: high-speed networks and high-performance end-user workstations (with enhanced processing power, main memory, and secondary storage). In addition, it should allow heterogeneity to percolate up to the application development level by supporting distributed applications that are developed by using more than one programming language.

Support for security and privacy. In most organizations, the security of data and authenticated access to their computing resources is paramount. Security within centralized environments is supported by limiting access through well-

defined access and authorization systems. However, within a distributed computing environment security remains a challenging problem, in part because the security of each resource is dependent on the security provided by the other components of the system.[17,18] If one component in the distributed system does not provide adequate security, the security of the entire system may be at risk.

Manageability. A distributed computing system consists of heterogeneous computing devices, communication networks, operating system services, and applications. The unavailability, incorrect operation, or inefficient operation of mission-critical devices, services, and applications could mean real losses to the affected organization.[19] Thus, for effective operation and management, these devices, services, and applications must be monitored and controlled.

Data access. Access to distributed, heterogeneous data sources creates a set of commonly identified user requirements. The requirements fall into two broad categories: connectivity and integration. *Connectivity* implies the ability to access data either at a remote site or stored by a data source that is different from the host source used by the application. *Data integration* implies the ability to utilize heterogeneous data in a seamless fashion so that the data do not intrude on the logic of the application. Integration should provide the translation of the data's schema, data types, data format, return codes, and error codes so that any heterogeneity in the underlying data sources is transparent to the user. Accessing and updating data at multiple heterogeneous sources must be transparent.

Support for role-specific transparency. It is generally accepted that some level of transparency is desirable in a distributed environment. It is also clear that the level of transparency desired may vary among the different users of the system. Transparency allows the application developer to view the system as a set of logical resources, alleviating the need to deal with the heterogeneous and distributed nature of the available resources. Thus, transparency is important in hiding details of the underlying systems and is instrumental in avoiding some of the complexity often associated with the development of distributed applications. The detail that an application developer may have to cope with may be too great for an end user interested in composing or customizing component applications to form a new application. In contrast, a system administrator may need access to much greater detail when trying to discover a performance problem. The environment must provide a vision of and control of the distributed system in some circumstances and provide transparency in others.

Support for visualization. Whether one is designing, developing, managing, or utilizing a distributed system, there is a large amount of information that a human must process. Visualization techniques can be used to verify, understand, and interpret this vast amount of information.[20] Once again, given the need to support a human-oriented computing environment, visualization techniques are required to facilitate a user's understanding and manipulation of this information.

Support for application development languages and tools. Programming distributed applications requires the use of third- or fourth-generation programming languages with appropriate extensions to allow effective use of underlying services. Languages such as Concert/C[21–23] have concurrent programming primitives that exploit underlying services in a more abstract way. Tools are emerging that help application programmers partition existing applications into the peer-to-peer or client/server paradigms.[24,25] Because new technology will continue to emerge, there is a clear need for a flexible "workbench" technology into which new tools can be added and work together with other new or existing tools. This technology will become particularly important as these tools become application- or domain-specific and are in turn combined by domain specialists to produce applications in other areas. This workbench must also provide the configuration management and version control tools to help application developers working in the new distributed environment to build, tune, and deploy these applications. Ideally, one could extend existing tools to accommodate distributed applications and, thus, assist in adaptation and reduce training costs.

Support for distributed debugging and testing. Distributed applications may generate large numbers of "messages" between the components of an application and with other applications. Distributed applications will not execute with any given *total order* of program events due to concurrency and asynchrony. Debugging such appli-

cations must enable a developer to trace events and faults, to capture error conditions, to identify interacting components, to replay events, and to map physical events to logical ones. Furthermore, new testing techniques must be developed since familiar testing techniques, such as regression testing, are impossible to use in the presence of asynchrony.

Accommodate evolving services. Underlying hardware and system services will continue to evolve, including the ones identified above. The computing environment must be able to accommodate a continuous process of extension, refinement, and standardization of services without having a negative impact on the applications developed in and supported by the environment.

Related work

As noted in the introduction, the focus of this paper is on the framework and architecture for services required of an environment for developing, deploying, and managing distributed applications. A great deal of research has been done on various aspects of distributed computing systems, from protocols and communication primitives, to fault tolerance, to distributed algorithms. A review of such work is clearly beyond the scope of this paper, though it is potentially relevant in the specification and definition of services discussed later in this paper.

Several efforts have, however, been addressing problems arising out of questions related to the nature of services and their integration and distribution in support of distributed applications. As previously noted, a primary difference between the CORDS project and other projects is the unique focus of CORDS on the peer-to-peer environment. Notwithstanding these differences and given the central issues raised in the previous section, there are many similarities in the goals of the projects and, therefore, in the types of services provided.

In this section, we provide a brief survey of other research projects that have addressed distributed services in architectural contexts. Within these projects, various terms are used: architecture, framework, etc. In an effort to relate these efforts to our own, we attempt to use a single set of terms; these are defined in Table 1.

Table 1 Basic terminology

Model	An abstraction of real-world entities into a set of precisely defined concepts, and relationships. Examples include a computational model, data model, communication model, and a naming model.
Reference model	A general model defining terms, concepts, and relationships in a particular area that can be used by other, more specific models or used as a vehicle for comparison of models in that area. Examples would include the OSI reference model and the ODP reference model.
Computational model	An abstraction of real-world entities into a set of precisely defined computational concepts and relationships. Examples include the process model,[26] relational model,[27] and CSP.[28]
Framework	The definition and organization of concepts to satisfy a set of requirements for a system. Examples include the OSI management framework and the Internet management framework.
Functional framework	The definition and organization of logical services and functions that satisfy a set of requirements for a system. An example would be the CORDS functional framework presented later.
Architecture	A refinement and specification of a functional framework in terms of one or more models.
Environment	Instantiation of one or more architectures and possibly other things such as tools or languages that do not have an architecture or have an unspecified architecture. ANSAware and OSF DCE are examples of such an environment.

Specifically, we discuss: the open distributed processing basic reference model (ODP),[29] the Advanced Networked Systems Architecture (ANSA),[1,2,30] UNIX International's ATLAS Distributed Computing Architecture,[3] X/Open Common Applications Environment,[4] the RACE Open Services Architecture (ROSA),[31–33] and the Multivendor Integration Architecture (MIA).[34]

Open distributed processing. The open distributed processing (ODP) standardization effort is aimed at developing standards to support open distributed processing within an enterprise framework.[29] The eventual standards are intended to enable enterprises to cope with heterogeneous systems and information sources. Work on the standards are done as Working Group 7 of the International Organization for Standardization (ISO) IEC Subcommittee 21, which is responsible for standards in information technology. Current work is focused on the development of a reference model for open distributed systems.

ANSA is an architecture for distributed computing.

The ODP standards are intended to support a broad range of distributed applications encompassing such areas as home entertainment, banking systems, medical systems, and information services, as well as others.

The reference model for ODP defines the technical basis for the ODP standards and specifies how ODP and its component standards relate to ISO reference models and existing standards. The reference model is composed of three parts:

1. A descriptive model that defines concepts that could be applied to any distributed processing system
2. A prescriptive model that presents a generic architecture for ODP
3. An architectural semantics that provides a formalization of the central reference model concepts

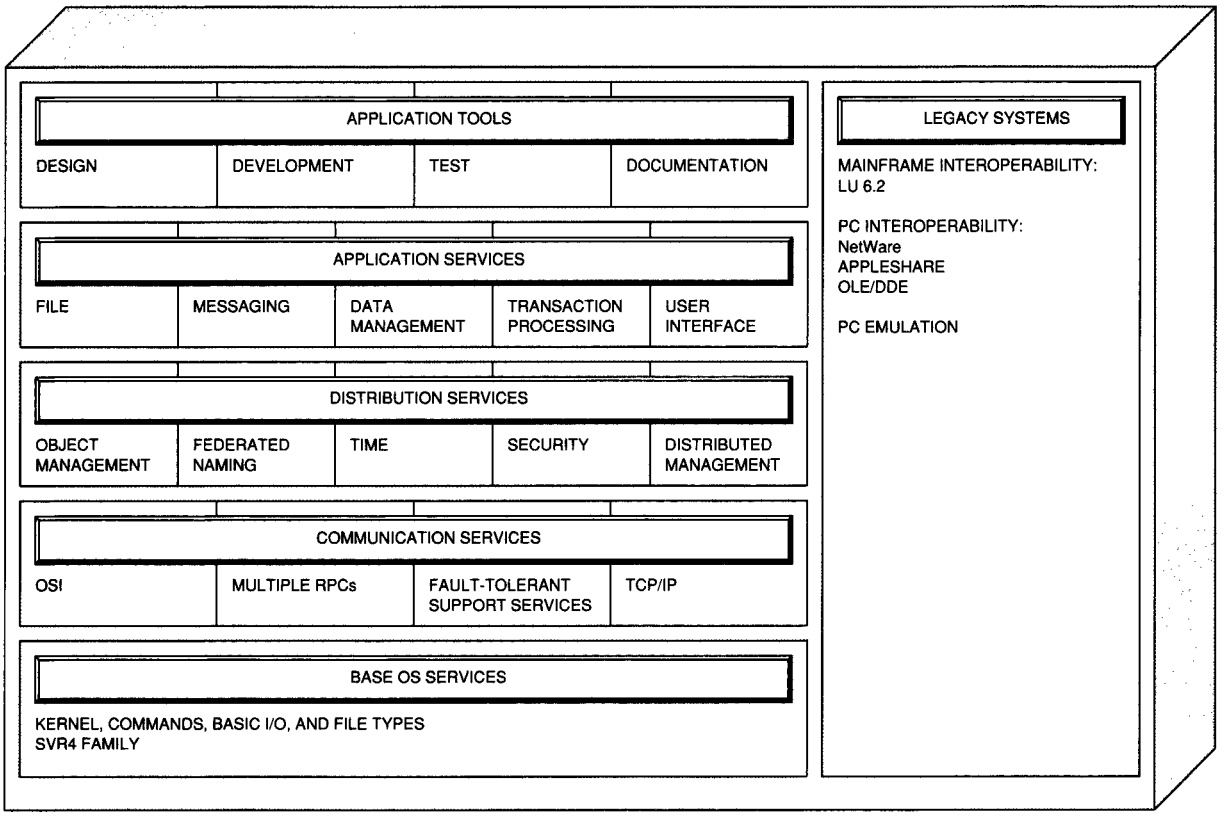
The standardization effort to date has focused primarily on the descriptive model and the formalization method(s) to be used to specify the architectural semantics. ODP has adopted the ANSA models described below.

ANSA. ANSA is an architecture for distributed computing. Its objectives include the integration of products from multiple vendors, scalability, and graceful evolution.[2] The architecture is based on a set of models, each offering domain-related concepts and rules: the enterprise model, information model, computational model, engineering model, and technology model. The enterprise model allows the construction of a model of an organization and its changes. The information model allows the designer to model the use of information. The computational model defines the facilities required of a programming system for the implementation of a distributed application. The engineering model defines the function of the infrastructure, and the technology model allows conformance rules for its realization. The ANSA "models" were used as the initial input to the definition of the ODP reference model and became the "viewpoints" of ODP.

In ANSA, all data are considered to be remote. It is assumed that one component does not have direct access to another. All data are accessed through remote procedure call (RPC), and all services are negotiated through trading services. Trading is one of the two important concepts introduced by ANSA. Trading is the process employed by clients to utilize attributes of a service, including locating appropriate servers and services on the network, a sort of "yellow pages" (telephone directory) access to services. The second concept, federation, allows interoperability between systems while allowing systems to maintain control of their domain. The emphasis of ANSA is on interfaces, particularly between services, and is based on the client/server model. ANSAware is a commercially available distributed computing environment based on ANSA.

UI-ATLAS distributed computing architecture. Unix International is a worldwide nonprofit consortium based in Parsippany, New Jersey. Their distributed computing environment is UI-ATLAS.[3] This project has three main objectives. First, to allow the computer industry to provide technology encompassing the widest range of interoperability of existing systems. Second, to see that the technology is provided at the lowest possible resource cost. Third, to have both the user and administrator see the entire system as one, single system.

Figure 3 UI-ATLAS distributed computing architecture



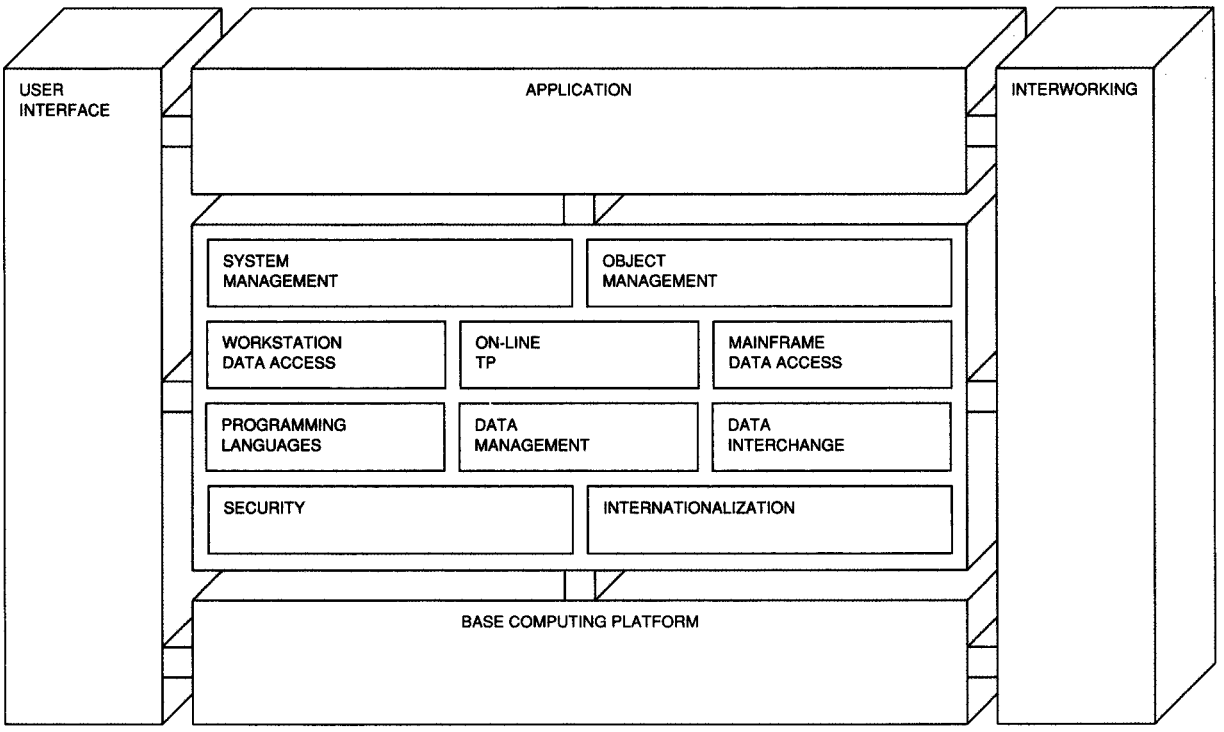
UI-ATLAS is a fairly comprehensive object-oriented architecture for distributed computing systems (see Figure 3). The vision of UI-ATLAS is of open, distributed computing made simple, consistent, scalable, robust, and manageable. This vision includes hiding system complexity. Overall, the top-level requirements are: integration, scalability, flexibility, extensibility, openness, information integrity, and security. UI-ATLAS will offer a superset of OSF DCE. In particular, it will extend transaction processing, database access, and integration with legacy systems.

X/Open Common Applications Environment. X/Open is a consortium of information system suppliers, user organizations, and software companies. X/Open has defined a service environment, the *Common Applications Environment* (CAE), (see Figure 4) which it describes as a "comprehensive and integrated system environment."

The goal of the CAE is to provide an "open systems" environment. To accomplish this goal, it defines a set of (implementation-independent) service interfaces. Thus, users and developers can develop applications that are portable and interoperable. The portability of applications is at the source-code level. The adoption of applications and services that adhere to the CAE specifications allows a heterogeneous mix of computer systems and application software. The specifications are developed by extending current systems (e.g., the UNIX** operating system) to provide a comprehensive application interface. All members of the consortium agree to support the defined service interfaces collectively known as the X/Open System Interface (XSI). In this way X/Open hopes to achieve "openness."

One of the primary concerns of X/Open is the selection and adoption of standards: *de jure* standards if they exist, *de facto* standards otherwise.

Figure 4 The X/Open Common Applications Environment



In the latter case, X/Open will try to obtain formal standards based on the chosen *de facto* standards.

ROSA. The RACE Open Services Architecture (ROSA) is an object-oriented architecture for integrated broadband communications (IBC) services.[32,33] Its goal is to provide a set of concepts, rules, and recipes for the specification, design, and implementation of "open" services. These services should be able to accommodate both new services and existing services and allow the interoperability of new and existing services. Furthermore, the architecture is to be independent of new and evolving network technologies.

Two basic frameworks are in the architecture: the Service Specification Framework (SSF) and the Resource Specification Framework (RSF). The concepts, rules, and recipes of the architecture are embodied by the SSF in a convenient form for a service designer. In ODP terminology, the SSF covers the computational viewpoint. The concepts for the SSF can best be described by the following object types that it defines:

- *Service control* allows a user to join, leave, suspend, and resume activities in a service, and to negotiate service parameters.
- *Session* allows the service control object to add, as well as change and delete user state information.
- *Charge* allows charges incurred during a user's session to be recorded and manipulated.
- *Transport control* maintains status information with respect to transport connections.

The RSF is oriented more toward system designers. Its purposes include defining abstractions for telecommunication resources, defining components required in target systems to fulfill openness requirements, and defining rules for extensions of its concepts. The SSF and the RSF are somewhat related through a requirement-mechanism relationship, the basis of the mapping between the frameworks (see Figure 5).

Implicit or explicit requirements of the SSF are put on the infrastructure through service specifications. New services are specified using SSF components, which helps to provide technology independence. Likewise, new network infrastructures are specified according to RSF rules, and thus do not impact the SSF.

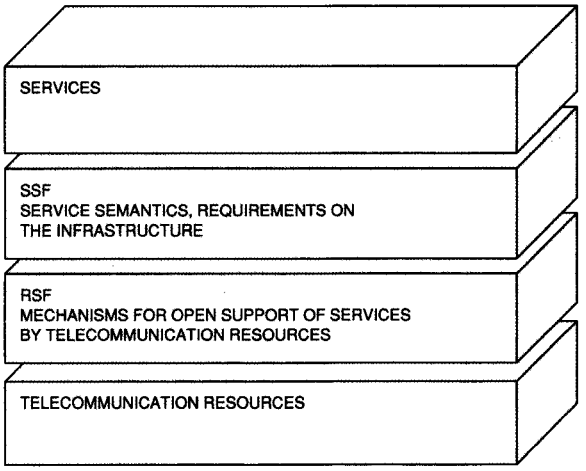
The following is a list of some of the suggested RSF objects.

- *Trail* provides multipoint transfer of multimedia end-user information. In addition, there are objects such as addMedia, removeMedia, suspendMedia, resumeMedia, and syncMedia.
- *X-connection* provides multipoint transfer of a monomedia end-user interface.
- *Creator* allows the creation and deletion of objects.
- *TypeManager* allows one to add and delete types as well as to find a list of subtypes that are defined.
- *Trader* maintains a dynamic information repository of services currently available in the system.
- *Clock* is for objects that require the current time or wish to receive regular interval "ticks."
- *Cluster* can manipulate and group a collection of objects.
- *Binder* can set up or destroy communication channels between objects.
- *Storage* provides storage and retrieval for passive clusters.

Multivendor Integration Architecture. The Multivendor Integration Architecture (MIA) is designed to allow the interoperability of and portability across heterogeneous systems based on the client/server model. MIA specifies a set of service interfaces that reflect the open systems architecture concept. MIA was defined by the Nippon Telegraph and Telephone Corporation (NTT).

The main goal of MIA is *multivendorization*—constructing a system with components from different vendors. The potential problems in multivendorization can be placed broadly into three categories: portability, interoperability, and procedures. To avoid these problems, it aims to establish a framework of standard interfaces for those services that most directly affect the user. MIA uses *open systems technology*: national and international standards, *de facto* standards, and specifications provided by open systems vendors.

Figure 5 Relationship between the SSF and RSF in ROSA

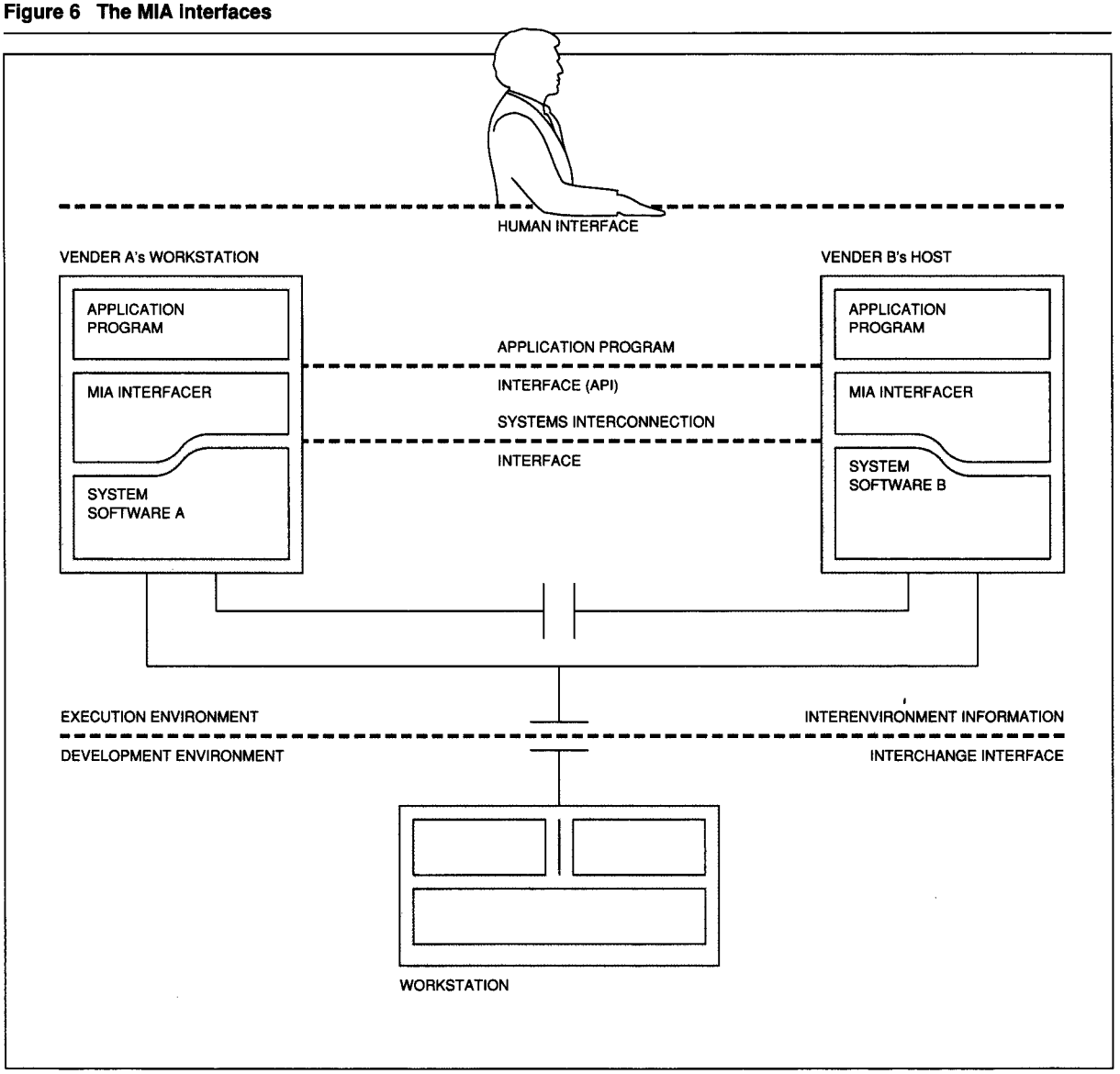


It is primarily intended as a guideline for vendors making NTT bids. NTT is not promoting MIA as a standard, though they are trying to incorporate relevant standards and to work with standards bodies on those interfaces that are not yet standardized.

The MIA defines a set of four interfaces depicted in Figure 6. The application program interface (API) is between applications and "basic software," e.g., international standards for COBOL, C, SQL (Structured Query Language), and is designed to allow application portability. The systems interconnection interface defines communications protocols and relies on both OSI and Internet protocols. The human interface defines display formats and workstation operations. A fourth interface, the interenvironment information interchange interface is defined to allow information to be passed from a development environment to an execution environment or to exchange application source code and data among execution environments. This interface defines interchange character sets and codes. The interface enables three types of development information to be passed: application program source code, database definitions (SQL data description language, DDL), and screen definitions.

In addition to the interfaces, NTT has defined eight conformance classes for the MIA specifications.

Figure 6 The MIA Interfaces



Thus, vendors who wish to label their product as conforming to a particular class of MIA specifications must demonstrate it against the relevant test suite.

CORDS architecture

The view of distributed applications interacting in a peer-to-peer manner has led the CORDS project to adopt the process model originally described by Strom et al.[26] The nature of a process-oriented peer-to-peer environment for distributed applications is the central focus of the CORDS research project. Issues and questions regarding how to realize such an environment, what services it requires, and how existing systems can be incorporated and evolve were central in the research.

Some of this research has resulted in the development of the CORDS architecture.[35,36] The prime constituents of the architecture, namely the process model and the CORDS functional framework,

are described in this section, the former representing the abstract computational model used as the basis for the peer-to-peer view of computing, and the latter encompassing the underlying distributed services. This architecture is evolving as our experience with ongoing research prompts us to refine its definition. The partial realization of this architecture and the incorporation of tools, such as the distributed debugger,[12] have provided the basis for other research into the design, development, and management within a distributed computing environment. For an in-depth description of the CORDS architecture, see References 35 and 36.

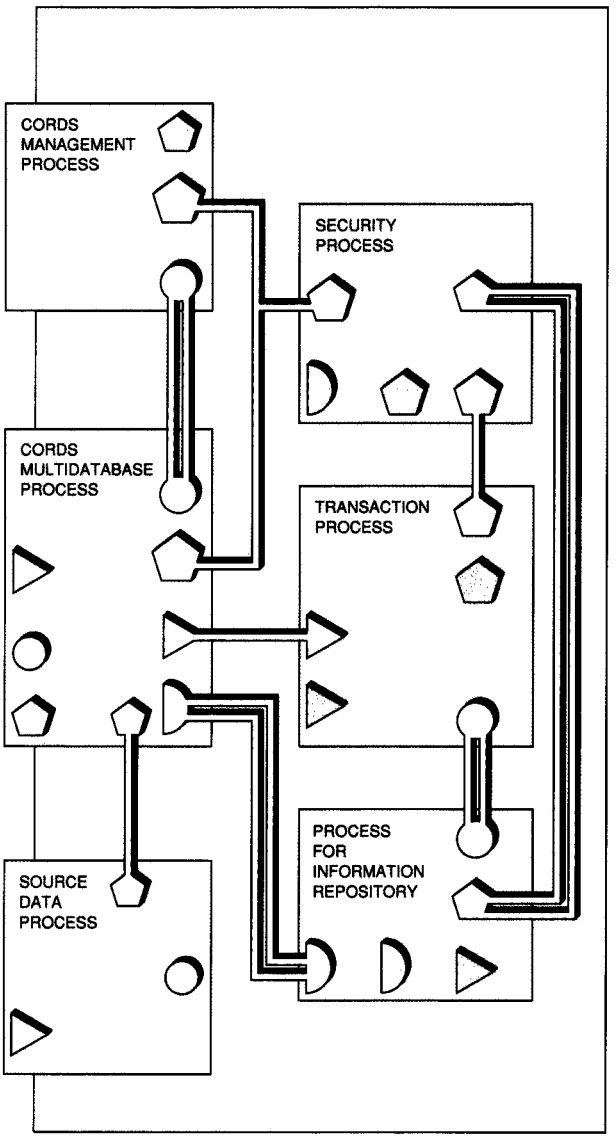
The process model. The process model provides a simple and elegant paradigm for building software. The model enables a distributed computing environment with consistent access to applications, data, resources, and services and facilitates the separation of logical, or application concerns, from the details of how services provided by the architecture are realized. This allows one to structure distributed software systems using processes as the building blocks. Each process is based on the concepts of encapsulation and information-hiding as well as on serial computation.

The independence of processes allows them to operate as peer entities. The process paradigm assumes an infinite universe; that is, there is no notion of a global state or a global time. Therefore, processes do not observe a given absolute order when executing distributed events. Each process maintains some local state, and only the program executing in that process can manipulate that state. All data are local to a process, and there are no shared variables; this is useful in simplifying the complexity of distributed applications. The model also assumes data and process persistence, i.e., assumes that these can be provided automatically by underlying run-time services if desired.

An active process interacts with another process by creating a channel on which it can send messages. A message channel is realized by connecting an output port of the sender to an input port of the receiver; each port is typed. The interface to a given process is determined solely by the types of its input ports. Thus, any processes with matching output and input ports may be connected if they choose. The type definition for a port can be written separately and independently of the internal process information. More details of the process model can be found elsewhere[26] as well as a comparison of the process model to object-oriented approaches.[37] Several languages implementing the process model concepts have been prototyped and studied.[38–40]

Figure 7 illustrates the peer-to-peer process model view of the CORDS multidatabase component and the CORDS management component. The

Figure 7 Utilization of the process model in CORDS



various shapes represent the typed message ports of the process model. Such ports may allow bidirectional, many-to-many communication as depicted in this figure. Some ports in the diagram are not connected, since all defined ports do not need to be utilized by every peer application. This is depicted by unfilled port symbols. These processes instantiate the CORDS multidatabase component[6] and the CORDS management component.[36,41] The services represented by these processes are defined in the CORDS functional framework described below.

Layers of the CORDS functional framework illustrate the logical separation of functionality among the various services.

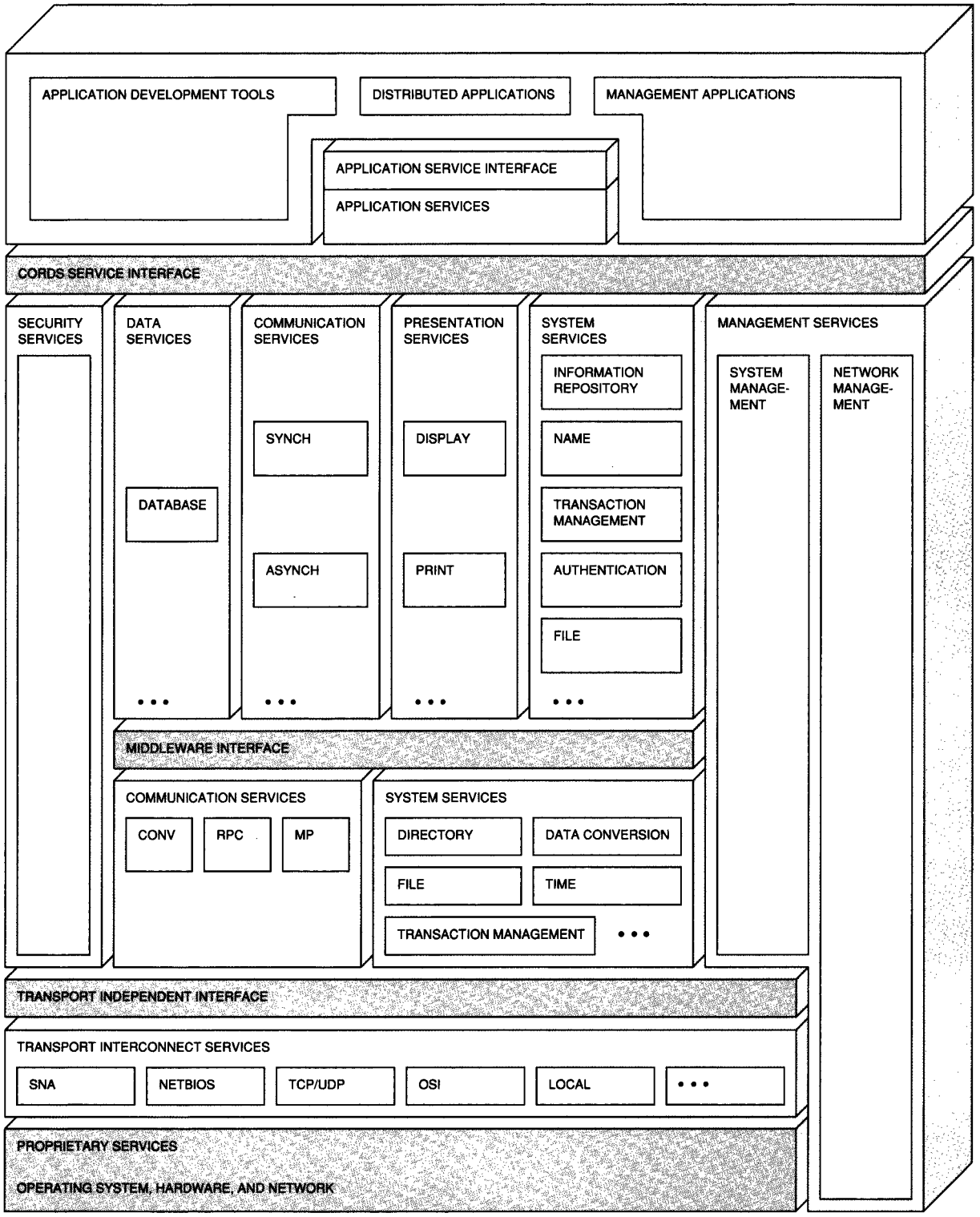
The CORDS functional framework. We now proceed to define the CORDS functional framework, depicted in Figure 8. This framework describes the organization of the logical services and functions that address the requirements identified earlier. Extensions to existing tools or the creation of new tools are required in order to satisfy some of those requirements, e.g., language and run-time extensions to support the development, debugging, etc., of distributed applications. What is important in the context of the architecture, however, is the services that such tools require. Given the breadth of the requirements, it was not possible to thoroughly explore all of the required services in detail within the scope of the project; some components, e.g., security services, remain to be explored in depth.

The underlying view of computation provided by the process model proved useful in considering the allocation and separation of services among the various components. For example, it became apparent during the research that the management component would require access to name services and to information about resources within the distributed computing environment. These services, subsequently embodied as name services and information repository services (within the systems services components; see Figure 8), would also be required by other system services and by components at the application layer (namely, applications, application tools, and services and management applications). Further, the data access and storage services required by the name services and the repository services, e.g., for storage of information necessary for the management processes and for the performance data gathered, could be provided by the data services component. This iterative approach led to a simpler view of the distributed services and provided a much more consistent partitioning of services. This, in turn, led to the realization that the data services could rely on the management services to provide information regarding network traffic, host loading, etc., needed for decisions in query optimization. As our understanding of the various services evolves with subsequent research, the services may continue to evolve based on the process view.

The layers of the functional framework illustrate the logical separation of functionality among the various services. It is likely that the components of one layer will make use of the services of the components at the layer beneath, though this does not imply that such a relationship may be strictly client/server, nor that a component act strictly as a server, especially with respect to components at the same logical level. Each of the five logical layers of the functional framework is now discussed.

- Applications layer: This layer encompasses distributed applications developed for the end users of the distributed computing system and any support tools for the composition of applications. It also includes applications used by two classes of specialized users, those responsible for the operation and management of the distributed applications and the distributed computing environment, and those who develop distributed applications.
- CORDS service environment: This layer specifies the services required by applications and tools in the applications layer. These services hide the peculiarities of the middleware layer and provide a standard set of interfaces to ensure that applications and tools that utilize services in this layer may remain independent of changes in the lower layers. A partial list of services includes security services, data ser-

Figure 8 The CORDS functional framework



vices, communication services, presentation services, system services, and management services. The specification of the CORDS service environment (CSE), as well as its instantiation through prototypes, is one of the objectives of the CORDS project.

- Middleware: Standardization efforts (such as OSF DCE) are building platforms to hide the details of individual proprietary systems and to provide services across these systems. An objective of the CORDS project is to identify the services of this layer, to determine the completeness of existing middleware systems and proposals (see Reference 42 for a study of the adequacy of middleware services to support distributed application development environments), and to enhance these services where required.
- Transport interconnect services: This layer identifies the basic set of services required to connect heterogeneous systems. The interconnection services in the middleware employ services in this layer.
- Proprietary services: The base layer consists of the services provided by the proprietary hardware, operating system, and network services.

Two aspects regarding the description of the functional framework must be kept in mind. First, one motivation for using a *layered* framework is to facilitate the reader's understanding of the services and their interactions. It allows a clear representation of the services available to (or required by) each group: applications, tools, or services. Second, the CORDS functional framework presents a *logical* view of a system. It allows one to provide the user with the services required to design, build, and maintain distributed applications. The goal of this framework is to satisfy the requirements identified earlier. The use of services in any layer are not precluded by any other layer; those users who require lower-level services may utilize them.

Emerging technologies will provide a computing environment that differs from the present environment, in scale if not in concept, by such a large factor that many of the present approaches to system development and use may become obsolete. As a result, there are two implications for any framework for distributed computing that is to support long-term development. First, the framework must allow systems to evolve to take advantage of the new features and methods provided by these technologies. Second, it should allow systems to *span* a spectrum of technologies. We elaborate on the layers and services within the CORDS functional framework in the remainder of this section.

CORDS application layer. The CORDS application layer encompasses distributed applications developed for end users, applications used in the operation and management of the distributed applications and the distributed computing environment (management applications), and applications (i.e., tools) used by those who develop distributed applications (development applications). All of these applications may make use of applications developed as services for other applications (application services).

Application services are built upon the services provided by the CSE and may be used by other components in the application layer: application development tools, distributed applications, and management tools. An example of an application service would be a visualization service. The service, provided by one or more visualization tools, would include the ability to manipulate data visually, the ability to perform pattern matching on visual data, the ability to visually select portions of the data and filter the data, the ability to browse and edit the visual data, and finally, the ability to improve visual data such as graphs without distorting the information the data contain. The tools that make up the visualization service depend on the presentation services of the CSE.

Application development tools are used for the development of distributed applications. Application developers should be able to select the most appropriate tool for each required function and be confident that the chosen suite of tools can work together. This idea suggests that standard architectures for tool integration (both control and data integration) and intertool communication are necessary. New approaches, which provide a finer grained data exchange between tools, must be integrated with the existing tool architecture. This framework will itself make use of the underlying CORDS services.

Application development in a distributed environment requires languages to facilitate programming. Distributed debugging tools providing capabilities such as the monitoring of communications between processes within an application,

and the replaying of previous executions, will also be required. Tool or service mechanisms need to be provided for modeling and schema definition, access control, interprogram communication, and resource and execution service binding. Static and dynamic naming and registering of resources and their capabilities are needed.

The application development environment relies on data services, presentation services, communication services, and system services, particularly name, authentication, and transaction management services, of the CSE. The CORDS application development architecture assumes that all industry-standard tool services can be provided by an interface to the underlying CORDS services. No assumptions are made, at present, about the distributed programming model or language necessary to create distributed applications.

Management tools are used for the management of distributed applications, system services, network services, and resources. Examples of management applications are modeling and simulation tools, monitoring and control tools, and analysis and report tools. Modeling and simulation tools are used to model complex application, system, and network configurations and determine "what-if" performance for those being developed. Monitoring and control tools are used to keep track of the behaviour of managed entities and to perform control actions when needed. Analysis and report tools are used to perform analysis (such as statistical analysis) on the monitored data and produce useful reports for the systems and network administrators. These management tools make use of the services provided by the CSE, particularly the management services.

Distributed applications are executed by end users; run-time support for such applications is provided by underlying services.

The CORDS service environment layer. Our goal is to define and elaborate an environment for designing, developing, and managing distributed applications in a peer-to-peer environment. The CORDS service environment consists of those services required to support application development tools, distributed applications, and management tools. To fully specify the functional framework of the CSE, one must define:

- The CORDS service interface, the services available to applications and tools
- The components and component relationships within the CSE, that is, the services provided by each component
- A mapping from the services specified in the CORDS service interface to the components within the CSE
- A mapping from the services required by components within the CSE to the services specified in the middleware interface

In this paper, we have concentrated on the set of services to support distributed applications and tools. The services are grouped into components and logical collections of subcomponents. The subcomponents do not necessarily partition the functionality of a component. Thus, within a single component, two subcomponents may provide overlapping services. This description of the CSE represents our understanding of the needs and the information required. Ongoing research is aimed at assessing and validating these services and their relationships. Within the scope of the CSE, several assumptions are made:

- Each service component may be composed of subcomponents that may be distributed services.
- Each service should be considered a black box. Thus, changes *within* a service component should not affect other components, tools, or applications, allowing the (future) migration of environments and applications, which are based on the CORDS framework, to incorporate new technologies.
- A component may represent a number of services, each of which may have its own specific interface. The component interface would be the union of the individual service interfaces.
- Each component is assumed to have a *management* interface, an interface that can be used to collect component-specific information about its state, performance, operation, errors, events, etc. The information collected via this interface is in addition to any other reporting the component may do, such as a return code.
- The process model will be employed to model the components and their interaction in the architecture.

We now describe each of the CSE components in more detail.

Data services—Support for distributed data services within a distributed computing environment is essential. The data services component encompasses several data sources within a single logical umbrella. The database service offers the standard functionality of database management systems. The current service includes, but is not restricted to, navigational, relational, and object-oriented databases and multidatabases. The multidatabase provides a single logical view of multiple, heterogeneous data sources that are distributed throughout the computing environment. The multidatabase subcomponent is described in greater detail elsewhere.[6] Other services will be added within the data services component as needed, for example, an object store.

The data services use the name service, transaction management service, and information repository service provided by the systems services, as well as the communication and security services.

Presentation services—Presentation services provide the functions required to display information. The services satisfy several characteristics: the ability to present various kinds of data—plain text, typeset text, graphic images, video, audio; the ability to use various kinds of devices—workstations, printers, high-resolution display units; the ability to handle input events; and the ability to provide access that is local or distributed. This set of services will continue to be refined to meet evolving graphics standards, such as GKS[43] or PHIGS,[44] and to incorporate new services (e.g., multimedia) as the technology becomes available.

For example, the X Window System** permits a display device to be connected with an application running on a remote host. It is currently done by explicitly specifying network addresses. In the future, such information could be extracted from the name service, thus alleviating work by the toolkit builder, system manager, application developer, or application user.

Management services—A critical aspect of a distributed computing environment will be the ability to configure, monitor, and control a wide range of applications, services, networks, and devices (which we collectively call *managed objects*). Information about the managed objects will be needed by management tools. Current activities in network management will provide techniques and tools for specifying and collecting network management information. However, at higher levels, the collection of information about system services and applications, tools to analyze the information, and services to monitor and control system activities will be needed.

Presentation services provide the functions required to display information.

The management services consist of several subsystems: management information repository subsystem, configuration subsystem, monitoring subsystem, control subsystem, and management agents. The management information repository subsystem consists of a set of information repositories providing storage for management information. The configuration subsystem is responsible for keeping track of the configuration information on managed objects, and for initiating and terminating managed objects. The monitoring subsystem is responsible for monitoring the behaviour of managed objects. The control subsystem performs appropriate control actions on managed objects as a result of their behaviour being monitored by the monitoring subsystem. The management agents are responsible for monitoring and controlling the behaviour of managed objects on behalf of users (or management tools). The CORDS distributed management architecture and details on its components are described in greater detail elsewhere.[45]

Management services use services from the data service, security service, name service, and communication service. Management agents depend on services that may be specific to particular networks, operating systems, or hosts. We assume here that such agents are provided to the management service components as closed units along with descriptions of what they provide, how they may be invoked and collected, and where they are applicable.

Communication services—The development of distributed applications and tools requires access to communication primitives that enable data and control information to flow between components. One of the objectives of CORDS is to explore the use of the process model in the architecture as a simple (logical) mechanism to allow applications to communicate. This mechanism may be mapped to more complex communication mechanisms at a lower layer, transparent to the application developer or user.

Two basic forms of communication are required: synchronous and asynchronous. Both types of primitives could have several realizations using services provided by the middleware layer. Initially, the services may be provided by an RPC-like mechanism, presumably independent of any specific RPC implementation. Eventually, such exchanges should take place on a peer-to-peer basis.

System services—A variety of information about the distributed system is needed to operate within the system and to ensure that it performs efficiently. Some of this information will be required by distributed applications, whereas other information may be needed by management functions.

Services are provided by components within the logical collection of system services and include naming services (*directory*), transaction and recovery services (*transaction management service*), authentication and security services (*authentication service*), a repository service (*information repository service*), and file, operating, and run-time services. Some of these services, especially the last three identified, may actually be provided by existing services at the middleware layer or proprietary systems. The inclusion of such services within the CSE is to (1) provide a logically consistent view of available services within the CSE and for processes at the application layer, and (2) provide the means to incorporate future functionality or provide a single interface to multiple realizations of the services at the middleware or proprietary systems layers.

Comparison of architectures. A detailed comparison of the CORDS architecture and the other architectures and frameworks discussed earlier is beyond the scope of this paper. Nevertheless, some general comments comparing CORDS to the other efforts are in order.

CORDS takes a peer-to-peer view of computing rather than the client/server view of such others as ANSA, OSF DCE, UI-ATLAS, and MIA. As mentioned earlier, we believe that client/server computing will evolve to a peer-to-peer environment, and research is required to begin to understand the requirements and needs of such an environment and how that evolution can be smoothly accommodated.

Moreover, CORDS derives many of its requirements from the anticipated needs of the end user, application developer, and system administrator. The objective is to identify services needed by these classes of users and to identify what tools are required to simplify their tasks. This is one reason why the CORDS project has been concerned with services to support access to heterogeneous data sources, use of distributed transactions, application management, distributed debugging, and visualization.

CORDS also assumes the existence of middleware to provide basic services across heterogeneous computing platforms. OSF DCE and ANSAware represent such middleware. One objective of the CORDS architecture was to hide details of the middleware from application developers and to insulate them from changes in middleware as platforms evolve. Middleware, such as OSF DCE, provides basic platform interoperability service, but broader sets of services are also required to support applications and tools.

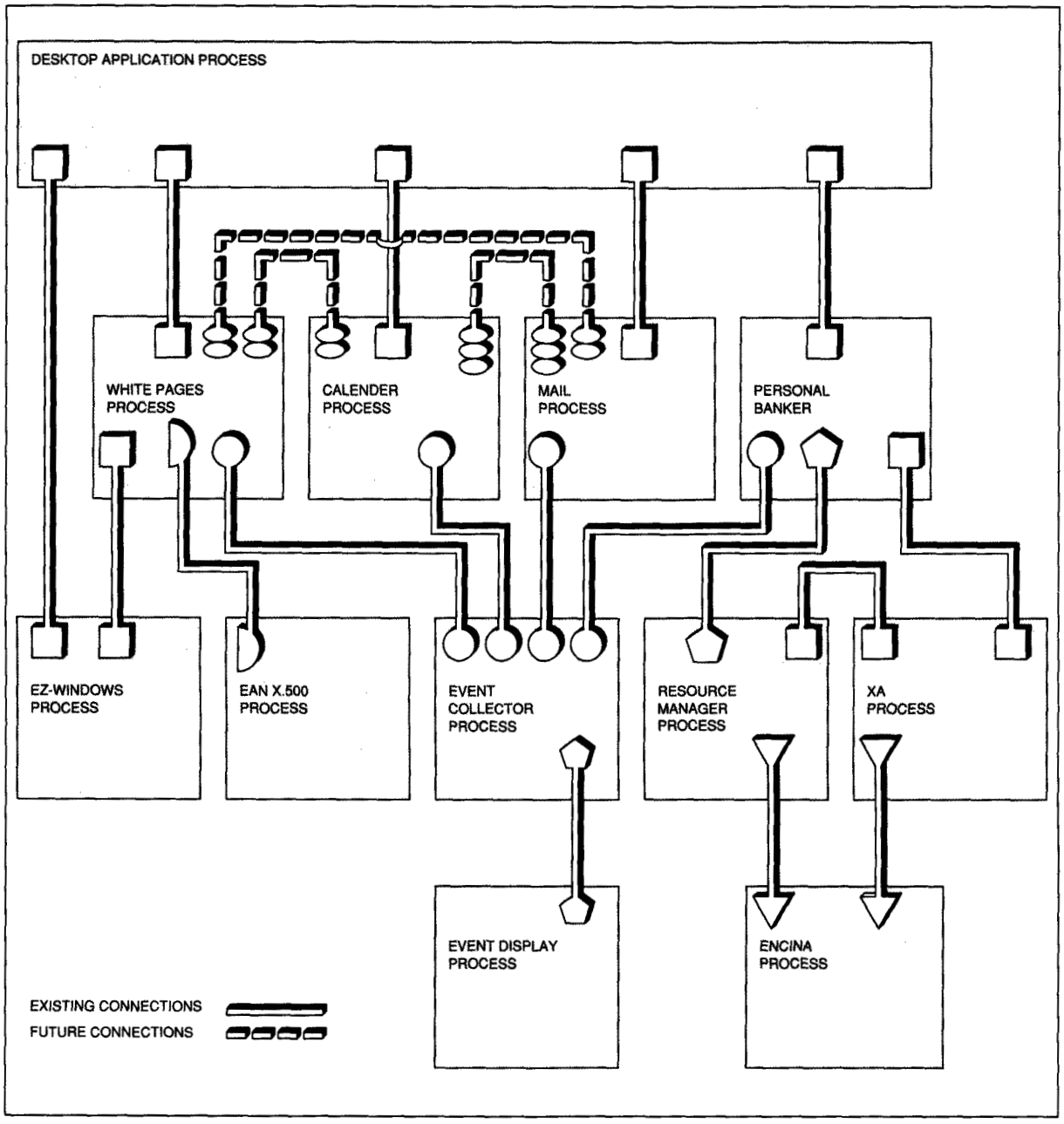
The ROSA effort takes a similar approach to that adopted by CORDS, although the focus is on broadband telecommunication services. Thus, although there are similar objectives such as scalability and openness, the computing and communication environments and end-user communities are different.

Finally, the ODP reference model represents a single collection of concepts and terms for describing distributed computing systems. It should be possible to map the various distributed computing architectures, frameworks, and environments discussed in this paper, including CORDS, to the reference model to examine similarities and differences. This comparison, though interesting, is beyond the scope of this paper.

Validating architectural concepts

A team of researchers, developers, and graduate students developed a prototype system[46] to eval-

Figure 9 Prototype process interactions



uate our preliminary ideas about the architecture. The development took place at the IBM Centre for Advanced Studies (CAS) during the summer of 1992. The prototype included basic services, similar to those described in the previous section, and a simple test suite of applications that utilized these services. The prototype development took place part way through the project, and many of our ideas about the functional framework were in the formative stages. One objective of the proto-

type experience was to have these ideas evolve and mature.

Basic services included a process communication facility that embodied the essential concepts of the process model and other services required by a prototype distributed application. The application integrated components for telephone directory white pages, electronic mail, a calendar system, and a personal banking service (similar to a personal automatic teller machine). In this section, we present an overview of the design and development experiences of this effort.

Prototype design. The prototype components can be divided into two broad categories that reflect its design: services (systems and applications) and applications. Figure 9 depicts a process-oriented view of the relationships among the various prototype components. Not all components need to be active simultaneously, and connections can be established or terminated dynamically. Moreover, these applications represent those of a single user; other mail processes, for example, could exist for other users. The logical organization of these components within the CORDS functional framework is illustrated in Figure 10.

Services. To provide distributed process control and communication primitives in a single homogeneous infrastructure, a library of routines was designed to provide process model primitives for the application programmer. A virtual distributed process space was designed and implemented to support virtual processes that spanned heterogeneous computers running OSF DCE. The design used a communications library to implement the process space instead of a language, such as Concert/C[47] or Hermes.[39] These communication services (Process Comms. in Figure 10) could be used, in addition to the OSF DCE communication primitives, to create and communicate with distributed (virtual) processes. These services are not explicitly depicted in Figure 9 since they are realized as the process-to-process connections.

A process server (Process Server) was implemented to provide the actual mechanisms to create, manage, and trace communications between processes in the virtual process space. It implements the process server concepts presented in Cygnus[48] from the University of Michigan, and Concert/C[47] from the IBM Thomas J. Watson Research Center.

Although OSF DCE included the GDS X.500 implementation, the prototype utilized the EAN X.500[49] (EAN X.500) version. The reason for choosing this X.500 implementation was that subsequent research developments required an X.500 service that understood transaction semantics. Our plan was to achieve this by including transaction facilities within the EAN X.500 service.

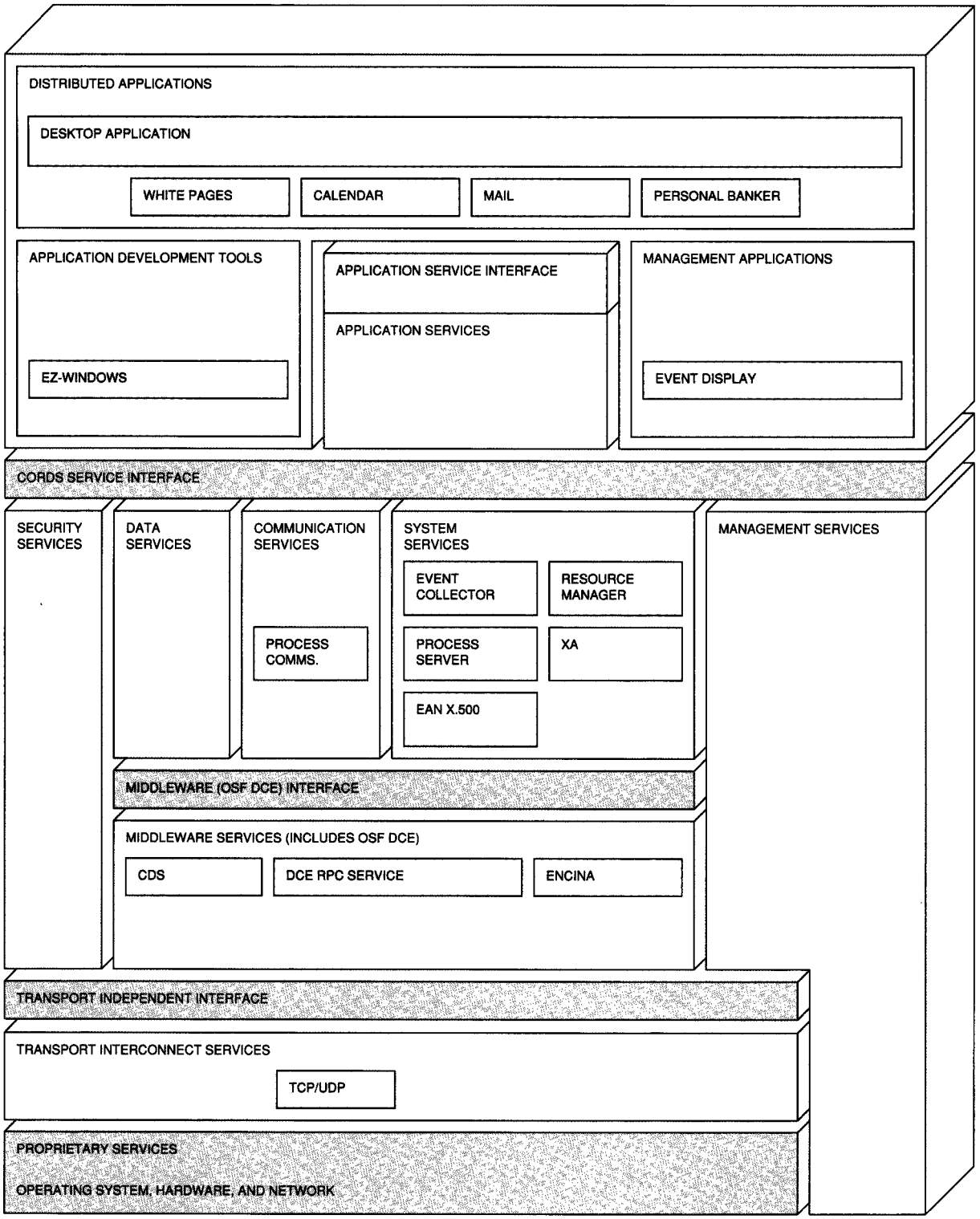
Prototype components can be divided into services and applications.

To provide the transaction management functionality lacking in OSF DCE, the project adopted the X/Open distributed transaction processing system, XA[50] (XA). This system defines a protocol between resource managers (Resource Manager) and transaction managers (Encina) to provide global control of distributed transactions. To expedite the development of resource managers, the project implemented an XA-interface bridge using Encina[51,52] to perform many of the required functions, including data recovery.

The EZWindows (EZ-Windows) system developed at IBM was used to facilitate GUI development. It provided a higher-level language for dynamically constructing Motif** windows and reduced the need for arcane X Window System programming.

The event collector[12] (Event Collector) collects communication events from an RPC monitor as well as the communications monitored by the process server. Servers and clients using the DCE RPC were developed in the usual fashion. However, in addition to the usual configuration steps, the developer arranged to have the output of the OSF DCE Interface Definition Language (IDL) compiler passed to a postprocessor. The postprocessor automatically instrumented the RPC client and server to send communication event messages to the event collector. Events are displayed on event time lines representing running processes. As illustrated in Figure 9, events from all of the basic

Figure 10 Prototype components and services within the CORDS functional framework



applications were sent to the event collector. These events were then displayed via the event display process (Event Display). Although the event display is placed within the management applications of the application layer, it could also be placed within the application development tools since it proved to be valuable in tracing and debugging interprocess communication.

The logical organization of these components also illustrates services relied upon at the middleware level. These services were primarily those provided by DCE with additions as needed, such as Encina.

Prototype applications. A suite of applications for a distributed office environment was built and included electronic mail, appointment scheduling, telephone directory white pages, and a personal banker. Where possible, the project took existing applications and re-engineered them for the distributed environment. Re-engineering allowed us to assess the effort and complexity involved in adapting legacy applications to the services within the CORDS functional framework.

The mail system is based on an X Window System version of the RAND message handling system.[53] The new mail system was decomposed into a user interface component and peer message transfer agents, which communicated by using virtual process communication primitives.

The project re-engineered a personal calendar program developed at the IBM Zurich Laboratory. The program was decomposed into two parts: a client with user interface and a personal server that managed an individual calendar. The decomposition made it possible to add a new feature, a meeting scheduler. A user wishing to schedule a meeting would use the client and contact the (personal) servers associated with each of the people involved in the meeting. The calendar system also used the communication libraries for communication between the client and server.

A white pages client providing information about the project and CORDS participants was developed. It used EZWindows to build the user interface, and the communication libraries were used for communication with the EAN X.500 server. As expected, EZWindows considerably reduced the time and effort required for user interface development, and communication tracing made it easy to monitor the X.500 usage.

Finally, a personal banker, similar to an automatic teller application, was used to investigate the requirements transactions would place on the environment. Starting with an X Window System automatic teller demonstration from Encina, a number of bank account resource managers were constructed. We found that the XA-interface bridge made it easy to include new resource managers.

Experiences. The project aided our understanding and supported a number of key concepts in CORDS. First, the process communication primitives and the process server demonstrated the feasibility of the process model as a useful paradigm for developing distributed applications in a heterogeneous environment. Second, certain services, such as transaction support, were identified as requirements of applications and application services. These requirements helped to further refine the service framework of the architecture. Third, process communications and, in particular, the event tracing facility demonstrated the usefulness of providing system monitoring. Finally, valuable experience and knowledge of software interfaces, integration issues, and the practical implications of heterogeneity were gained during the implementation effort.

Concluding remarks

Our research into a distributed computing environment and support services was motivated by what we perceived as two eventual paradigm shifts. First, the trend toward more human-oriented computing suggests that future computing environments will have to provide the building blocks and composition mechanisms to enable domain specialists to build customized applications. This trend requires development environments in which underlying services and platform details are hidden, in which distribution is transparent, and in which operation and management can tailor and optimize run-time behaviour. It also implies that methodologies are required to facilitate composition and creation of application toolkits, components, and building blocks along with interconnection mechanisms.

Second, we see the emergence of client/server interaction as an interim step toward a broader computing environment based on peer-to-peer interaction. As with the current client/server environment, new sets of tools, languages, and services will be required to facilitate the devel-

opment of applications in this environment. We also felt that a peer-to-peer model could accommodate existing applications—essentially wrapping each as an entity capable of (perhaps limited) interaction with other applications. Peer-to-peer computing is also consistent with the trend toward more human-oriented computing environments as just described. Application building blocks viewed as peers that can be interconnected is a simple model familiar to users who must deal with human peers on an everyday basis. Of course, providing the mechanisms to make such an interconnection of applications straightforward poses significant challenges.

On the basis of these trends, a number of requirements for future distributed computing environments were identified. Problems arising from trying to realize a computing environment satisfying these requirements have been the focus of the research within the CORDS project. One aspect of this work has been the identification of an architecture and framework for a distributed computing environment. The architecture has emerged in parallel with research on problems arising from some of the issues cited above and has evolved as our understanding of problems, issues, services, and dependencies has changed. An early version of the architecture was validated with a prototype that met with some success. The prototype experience also helped to clarify some ideas and to illustrate some of the complexities and challenges.

Perhaps more than anything the architecture has helped us to understand what problems exist, even if we did not have the resources to pursue them, and has provided a context for considering interdisciplinary problems arising in different areas of distributed computing, such as interactions among multidatabases, distributed application management, and system management. The interdependencies, interactions, and relationships among services in these different domains would not have been apparent had it not been for this broader view. The research in these areas, in turn, has reinforced the need for and potential of peer-to-peer interactions.

Though the project to date has addressed only some of the fundamental issues in the development of an environment based on and services supporting peer-to-peer computing, it has validated our original hypotheses and has met with several successes, including several prototypes. It has also demonstrated the need for interaction among experts in multiple areas of distributed computing. Since an operational distributed computing environment will entail many different services, it is imperative that dependencies among such services be understood and that the integration of such services be explored.

Acknowledgments

We would like to thank all the developers, faculty, students, researchers, and programmers who have worked on the CORDS project: Hasina Abdu, Utpal Amin, Gopi Krishna Attaluri, Joshua Auerbach, David Bachmann, David E. Bacon, Pravin Baliga, J. Michael Bennett, Jay P. Black, Gerold Boersma, Barry Brachman, Martin Brachwitz, Dexter P. Bradshaw, Lauri J. Brown, Yvan Cazabon, Jhitti Chiarawongse, Hsien-Kwang Chiou, Mariano Consens, Crispin Cowan, Kerman Deboo, Jan de Meer, Alexander Dupuy, Frank Eigler, Danilo Florissi, Patricia Soares Florissi, Arthur Goldberg, German Goldszmidt, Masum Hasan, Curtis Hirischuk, Yen-Min Huang, Yanni Jew, Michael Kalantar, Mike Katchabaw, Zenith Keeping, Willard Korfhage, Thomas Kuntz, Alex Liu, Ming-Ling Lo, Greg Lobe, Andy Lowry, Kelly Lyons, Andrew Marshall, T. Patrick Martin, Alberto Mendelzon, Brian Minard, Shahrokh Namvar, Gerald Neufeld, Manny Noik, John O'Neil, Ian Parsons, Glenn N. Paulley, Frank Pellow, Wendy Powley, Gary Promhouse, David Rappaport, C. V. (Ravi) Ravishankar, Jerome Rolia, James Russell, Jonathan Schaeffer, S. Sengupta, Avi Silberschatz, Mike Starkey, Rob E. Strom, Duane Szafron, Xueli Tang, Dimitra Vista, Sam Wang, Zhanpeng Wang, Gerald Winters, Weipeng Yan, Shaula Yemini, Yechiam Yemini (YY), Daniel Yellin, Jianchun Zhang, and Qiang Zhu. We apologize for any omissions or errors in this list.

We would also like to thank the referees of this paper for the excellent comments and suggestions for improving the paper.

Research reported in this paper was supported by the IBM Centre for Advanced Studies and by the Natural Sciences and Engineering Research Council of Canada.

**Trademark or registered trademark of Architecture Projects Management Ltd. (APM), Open Software Foundation, Inc., X/Open Company Limited, or the Massachusetts Institute of Technology.

Cited references

1. D. S. Marshak, "ANSA: A Model for Distributed Computing," *Network Monitor* **6**, No. 11, 3–22 (November 1991).
2. *Architecture and Design Frameworks*, Technical Report TR.38.00, Architecture and Projects Management Ltd., Cambridge, UK (February 1993).
3. "Unix Consortiums Building Distributed Computing Standards," *Database Reviews* (USA) **3**, No. 5, 4–7 (October 1991).
4. S. H. Dolberg, "X/Open in the 1990s," *Open Information Systems* **8**, No. 1, 3–19 (January 1993).
5. D. Fauth, J. Gossels, D. Hartman, B. Johnson, R. Kumar, N. Leser, D. Lounsbury, D. Mackey, C. Shue, T. Smith, J. Steiner, and W. Tuvell, *OSF Distributed Computing Environment Rationale*, Technical Report, Open Software Foundation, Cambridge, MA (May 1990).
6. N. Coburn and P.-Å. Larson, "Multi-Database Services: Issues and Architectural Design," *Proceedings of CASCON '92*, IBM Centre for Advanced Studies, Toronto (November 1992).
7. T. P. Martin, M. Bauer, N. Coburn, P.-Å Larson, G. Neufeld, J. Pachl, and J. Slonim, "Directory Requirements for a Multidatabase Service," *Proceedings of CASCON '92*, IBM Centre for Advanced Studies, Toronto (November 1992).
8. U. Amin, D. W. Bachmann, K. Deboo, and T. J. Teorey, "NESTMOD: The NetMod-NEST Interface," *Proceedings of CASCON '91* and Technical Report TR 74.064, IBM Centre for Advanced Studies, Toronto (October 1991), pp. 239–254.
9. G. Goldszmidt, S. Yemini, and Y. Yemini, "Network Management by Delegation—the MAD Approach," *Proceedings of CASCON '91*, IBM Centre for Advanced Studies, Toronto (October 1991), pp. 347–361.
10. J. W. Hong and M. A. Bauer, "Design and Implementation of a Generic Distributed Applications Management System," *Proceedings of the GLOBECOM '93*, Houston, TX (November 1993), pp. 207–211.
11. J. W. Hong, M. A. Bauer, and J. M. Bennett, "Integration of the Directory Service in the Network Management Framework," *Proceedings of the Third International Symposium on Integrated Network Management*, San Francisco (April 1993), pp. 149–160.
12. D. Taylor, "A Prototype Debugger for Hermes," *Proceedings of CASCON '92, Volume I*, IBM Centre for Advanced Studies, Toronto (November 1992), pp. 29–42. Also in *Volume II*, pp. 313–326.
13. T. Kunz and D. J. Taylor, "Distributed Debugging Using a Reverse-Engineering Tool," *Proceedings of the 3rd Reverse Engineering Forum* (September 1992).
14. K. A. Lyons, "Cluster Busting in Anchored Graph Drawing," *Proceedings of CASCON '92*, J. Botsford, A. Ryman, J. Slonim, and D. Taylor, Editors, IBM Centre for Advanced Studies, Toronto (November 1992), pp. 7–17.
15. M. P. Consens, C. N. Knight, and A. O. Mendelzon, *The Architecture of the G+/Graphlog Visual Query System*, Technical Report TR 74.054, IBM Canada Laboratory, 895 Don Mills Road, North York, Ontario M3C 1W3, Canada (April 1991).
16. M. Consens, M. Hasan, and A. Mendelzon, "Debugging Distributed Programs by Visualizing and Querying Event Traces," *Proceedings of the 3rd ACM/ONR Workshop on Parallel and Distributed Debugging* (May 1993); extended abstract.
17. B. Randell and J. E. Dobson, "Reliability and Security Issues in Distributed Computing Systems," *IEEE 1986 Symposium on Reliability in Distributed Software and Database Systems* (January 1986), pp. 113–118.
18. M. Satyanarayanan, "Integrating Security in a Large Distributed System," *ACM Transactions on Computer Systems* **7**, No. 3, 247–280 (August 1989).
19. C. A. Joseph and K. H. Muralidhar, "Integrated Network Management in an Enterprise Environment," *IEEE Network* **4**, No. 4, 7–13 (July 1990).
20. P. J. Finnigan and R. Marom, *Current Issues in Visualization*, Technical Report TR 74.100, IBM Canada Laboratory, 895 Don Mills Road, North York, Ontario M3C 1W3, Canada (1992).
21. J. Auerbach, M. Kennedy, J. Russell, and S. Yemini, *Interprocess Communication in Concert/C*, Research Report RC 17341, IBM Thomas J. Watson Research Center, Yorktown Heights, NY (October 1991).
22. J. S. Auerbach, *Concert/C Specification,* Research Report RC 18994, IBM Thomas J. Watson Research Center, Yorktown Heights, NY (1991).
23. J. Russell, *The Concert Interface Definition Language*, Research Report RC 19229, IBM Thomas J. Watson Research Center, Yorktown Heights, NY (1992).
24. A. C. Choi and W. Scacchi, "Extracting and Restructuring the Design of Large Software Systems," *IEEE Software* **7**, No. 1, 66–71 (1990).
25. H. A. Mueller and K. Klashinsky, "Rigi—A System for Programming-in-the-Large," *Proceedings of the Tenth International Conference on Software Engineering*, Raffles City, Singapore (April 1988), pp. 80–86.
26. R. E. Strom, "A Comparison of the Object-Oriented and Process Paradigms," *SIGPLAN Notices* **21**, No. 10 (October 1986).
27. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks" *Communications of the ACM* **13**, No. 6 (August 1970).
28. C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1985).
29. ISO/IEC/JTC1/SC21/WG7, *Basic Reference Model of Open Distributed Processing: Parts 1–5*, Technical Report CCITT X.901-X.905 and ISO 10746-1-10746-5, International Organization for Standardization, Geneva (1992).
30. *ANSA Reference Manual, Release 1.01,* Architecture Projects Management Ltd., Cambridge, UK (July 1989).
31. M. Key, "ROSA—RACE Open Services Architecture," *Seventh International Conference on Software Engineering for Telecommunication Switching Systems*, Bournemouth, UK (July 1989), pp. 16–20.
32. *RACE Open Services Architecture: ROSA Handbook*, Release Two, Technical Report D.WPQ.2 93/FTL/DNR/DS/A/013/b1, RACE Project, European Commission, Brussels (December 1992).
33. *RACE Open Services Architecture: ROSA Architecture*, Release Two, Technical Report D.WPY.2 93/BTL/DNR/DS/A/005/b1, RACE Project, European Commission, Brussels (May 1992).
34. *Multivendor Integration Architecture, Concepts and Design Philosophy*, Nippon Telegraph and Telephone Corporation, Tokyo (1992).
35. M. A. Bauer, G. Bochman, N. Coburn, D. L. Erickson, P. J. Finnigan, J. W. Hong, P.-Å. Larson, T. P. Martin,

A. Mendelzon, G. Neufeld, A. Silberschatz, J. Slonim, D. Taylor, T. J. Teorey, and Y. Yemini, *The CORDS Architecture: Version 1.0*, IBM Centre for Advanced Studies, Toronto (May 1993).
36. M. A. Bauer, G. Bochman, N. Coburn, D. L. Erickson, P. J. Finnigan, J. W. Hong, P.-Å. Larson, T. P. Martin, A. Mendelzon, G. Neufeld, A. Silberschatz, J. Slonim, D. Taylor, T. J. Teorey, and Y. Yemini, *The CORDS Architecture: Version 2*, IBM Centre for Advanced Studies, Toronto (1994), in preparation.
37. M. A. Bauer, R. E. Strom, N. Coburn, D. L. Erickson, P. J. Finnigan, J. W. Hong, P.-Å. Larson, and J. Slonim, "Issues in Distributed Architectures: A Comparison of Two Paradigms," *Proceedings of the International Conference on Open Distributed Processing*, Berlin, Germany (September 1993), pp. 411–417.
38. A. Black, N. Hutchinson, E. Jul, and H. Levy, "Object Structure in the Emerald System," *Proceedings of the ACM Conference on Object-Oriented Systems, Languages and Applications* (October 1986), pp. 78–86.
39. R. E. Strom, D. F. Bacon, A. Goldberg, A. Lowry, D. Y. Yemini, and S. A. Yemini, *Hermes: A Language for Distributed Computing*, Prentice-Hall, Inc., Englewood Cliffs, NJ (January 1991).
40. A. P. Goldberg, *Concert/C: A Language for Distributed C Programming—Tutorial*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY (March 1993).
41. J. W. Hong, M. Bauer, and M. Bennett, "The Role of Directory Services in Network Management," *Proceedings of CASCON '92*, IBM Centre for Advanced Studies, Toronto (November 1992).
42. J. Slonim, J. W. Hong, P. J. Finnigan, D. L. Erickson, N. Coburn, and M. A. Bauer, "Does Midware Provide an Adequate Distributed Application Environment," *Proceedings of the International Conference on Open Distributed Processing*, Berlin, Germany (September 1993), pp. 34–46.
43. *American National Standard for Information Systems: Computer Graphics—Graphical Kernel System (GKS) Functional Description*, ANSI X3.124.1 Edition, American National Standards Institute, New York (1985).
44. *Programmer's Hierarchical Interactive Graphics System (PHIGS)*, ISO/IEC 9592-1 Edition, International Organization for Standardization, Geneva.
45. M. A. Bauer, P. J. Finnigan, J. W. Hong, J. A. Rolia, T. J. Teorey, and G. Winters, "CORDS Distributed Management," *Proceedings of CASCON '93*, IBM Centre for Advanced Studies, Toronto (October 1993), pp. 27–40.
46. G. K. Attaluri, D. Bradshaw, P. J. Finnigan, N. Hinds, M. Kalantar, K. A. Lyons, A. D. Marshall, J. Pachl, and H. Tran, "Operation Jump Start: A CORDS Integration Prototype Using DCE," *Proceedings of CASCON '93*, IBM Centre for Advanced Studies, Toronto (November 1993), pp. 621–636.
47. S. A. Yemini, G. S. Goldszmidt, A. D. Stoyenko, Y. Wei, and L. Beeck, "Concert: A High-Level-Language Approach to Heterogeneous Distributed Systems," *Proceedings of the 9th International Conference on Distributed Computing Systems* (June 1989), pp. 162–171.
48. R. N. Chang and C. V. Ravishankar, *Language Support for an Abstract View of Network Service*, Technical Report, University of Michigan, Ann Arbor, MI (1989).
49. G. Neufeld, B. Brachman, and M. Goldberg, "The EAN X.500 Directory Service," *Journal of Internetworking Research and Experience* **3**, No. 2, 55–82 (June 1992).
50. *CAE Specification. Distributed Transaction Processing: The XA Specification*, X/Open Company Limited, United Kingdom (1991).
51. *Encina: Product Overview*, Transarc Corporation, Pittsburgh, PA (1991).
52. *Encina Toolkit Executive Programmer's Reference*, Transarc Corporation, Pittsburgh, PA (1991).
53. M. T. Rose, E. A. Stefferud, and J. N. Sweet, "MH: A Multifarious User Agent," *Computer Networks and ISDN Systems* **10**, No. 2, 1–26 (September 1985).

Accepted for publication March 30, 1994.

Michael A. Bauer *Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada (electronic mail: bauer@csd.uwo.ca).* Dr. Bauer is chairman of the Computer Science Department at the University of Western Ontario. He holds a Ph.D. from the University of Toronto in computer science. His research interests include distributed computing, distributed directories, and software engineering.

Neil Coburn *Antares Alliance Group Canada Ltd., Mississauga, Ontario L5N 1V8, Canada (electronic mail: nzca0 @amdahlcsdc.com).* Dr. Coburn completed his Ph.D. at the University of Waterloo in 1988. He worked as Research Assistant Professor in the Department of Computer Science at the University of Waterloo until 1993, when he joined Antares Alliance Group Canada Ltd. His research interests include multidatabases, parallel databases, query optimization, and the development and maintenance of large software systems.

Doreen L. Erickson *Department of Computer Science, Southern Technical University, Marietta, Georgia 30060-2896 (electronic mail: erickson@st6000.sct.edu).* Dr. Erickson received her Ph.D. from the University of Waterloo in 1993 and held a postdoctoral position with the University of Western Ontario and the IBM Centre for Advanced Studies in 1993. She is now Associate Professor of Computer Science at Southern Technical University. Her research interests include parallel and distributed computing and cryptography.

Patrick J. Finnigan *IBM Software Solutions Division, Toronto Laboratory, 1150 Eglinton Avenue E, Don Mills, Ontario M3C 1H7, Canada (electronic mail: finnigan@vnet.ibm. com).* Mr. Finnigan is a staff member at the IBM Toronto Software Solutions Laboratory. He received his M.Sc. in computer science from the University of Waterloo in 1994. His research interests include visualization for distributed applications and software engineering.

James W. Hong *Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada (electronic mail: jwkhong@csd.uwo.ca).* Dr. Hong is a research associate and an adjunct professor in the Department of Computer Science at the University of Western Ontario. He received his Ph.D. from the University of Waterloo in 1991. His research interests include distributed computing, operating systems, software engineering, and network management.

Per-Åke Larson *Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (electronic mail: palarson@maytag.uwaterloo.ca).* Dr. Larson is a professor in the Department of Computer Science at the University of Waterloo. He served as chairman of the department during 1989–1992. His research interests include multidatabase systems, query optimization and processing, and parallel databases.

Jan Pachl *SHL Systemshouse, Inc., Toronto, Ontario M5J 1R7, Canada (electronic mail: jkpachl@jeeves.uwaterloo.ca).* Dr. Pachl was a research staff member at the IBM Toronto Software Solutions Laboratory's Centre for Advanced Studies until 1993. His research interests include distributed systems and distributed algorithms.

Jacob Slonim *IBM Software Solutions Division, Centre for Advanced Studies, Toronto Laboratory, IBM Canada Ltd., 844 Don Mills Road, North York, Ontario M3C 1V7, Canada (electronic mail: jslonim@vnet.ibm.com).* Dr. Slonim is head of research for the IBM Toronto Software Solutions Laboratory's Centre for Advanced Studies. He received his Ph.D. from Kansas State University in 1979 and is an adjunct professor at the University of Western Ontario and the University of Waterloo. His research interests include databases, distributed systems, and software engineering.

David J. Taylor *Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (electronic mail: dtaylor@boomer.uwaterloo.ca).* Dr. Taylor is Associate Professor of Computer Science at the University of Waterloo. His research interests include distributed systems and software fault-tolerance.

Toby J. Teorey *University of Michigan, Ann Arbor, Michigan 48103-4943 (electronic mail: teorey@umich.edu).* Dr. Teorey is Professor of Electrical Engineering and Computer Science at the University of Michigan and is Associate Chair for Computer Science. He holds a Ph.D. from the University of Wisconsin in computer science. His research interests include data modeling, distributed databases, and network performance tools.

Reprint Order No. G321-5548.