The promise of distributed computing and the challenges of legacy information systems[1]

Michael L. Brodie
Intelligent Database Systems
GTE Laboratories Incorporated,
40 Sylvan Road, Waltham, MA 02254
brodie@gte.com

Abstract

The imminent combination of computing and telecommunications is leading to a compelling vision of world-wide computing. The vision is described in terms of next generation computing architectures, called Enterprise Information Architectures, and next generation information systems, called Intelligent and Cooperative Information Systems. Basic research directions and challenges are described as generalizations of database concepts, including semantic aspects of interoperable information systems. No matter how compelling and potentially valuable the vision may be, it is of little use until the legacy problem is solved. The problem of legacy information systems migration is described, in the context of distributed computing, and is illustrated with lessons learned from actual case studies. The basic research directions and challenges are recast in the light of actual legacy information systems. Recommendations for both realizing the vision and meeting the challenges are given, including the search for the elusive Killer Application and one fundamental challenge for future information systems technology.

Keyword Codes: H.0; H.1.1; H.4.0

Keywords: Information Systems, General; Systems and Information Theory, Information Systems Applications, General

1. World-Wide Computing

My professional goal is to contribute to making the world a better place by providing solutions to significant, practical problems (see Appendix). As a computer science researcher, this means that I want to produce the highest quality research and technology that is ultimately applicable to real problems so that the results are consistent with my beliefs. In this regard, I have high hopes and expectations for the potential benefits of world-wide computing. The vision is that problems or questions posed by one or more agents (e.g., humans or computer) can be solved as automatically and transparently as possible. Automatically means that the necessary computing resources [e.g., programs, information bases, information systems (ISs)] are identified and caused to interact cooperatively to effectively and efficiently solve the problem. Transparency means that all unnecessary details are not seen by the agent (e.g., locations and nature of the participating resources).

---

[1]An earlier version of this paper appeared in P.M.D. Gray and R.J. Lucas (eds.) *Advanced Database Systems: Proceedings of the 10th British National Conference on Databases,* Springer-Verlag, New York/Heidelburg, 1992.

In this section, I describe a world-wide computing vision in terms of cooperation amongst ISs augmented by a telecommunications vision that provides communication on a scale previously unthinkable by computer scientists.

1.1. The Vision

The vision of distributed computing is compelling. It says that soon the dominant computing paradigm will involve large numbers of heterogeneous, intelligent agents distributed over large computer/communication networks. Agents may be humans, humans interacting with computers, humans working with computer support, and computer systems performing tasks without human intervention. Work will be conducted on the network in many forms. Work task definition will be centralized (e.g., a complex engineering task) and decentralized. Tasks will be executed by agents acting autonomously, cooperatively, or collaboratively, depending on the resources required to complete the task (e.g., monitoring many systems of a patient or many stations in a factory). Agents will request and acquire resources (e.g., processing, knowledge, data) without knowing what resources are required, how to acquire them, or how they will be orchestrated to achieve the desired result. A goal of this vision is to be able to use, efficiently and transparently, all computing resources that are available on computers in large computer/communications networks.

1.1.1. Cooperative Work

Computers should support humans and organizations in their natural modes of thinking, playing, and working. Consider how complex activities are in human organizations such as a hospital (Figure 1). Each human agent (e.g., doctor, technician, nurse, receptionist) provides capabilities to cooperatively achieve a goal (e.g., improve the health of a patient). For a doctor to complete an analysis of a patient, the doctor may need the opinion of another doctor, the results of a laboratory test, and personal information about the patient. In general, the analysis is broken into sub-activities and appropriate agents are found for each sub-activity. Each sub-activity is sent to the appropriate agents together with the required information in a form that the agent can use. Cooperating agents complete the sub-activities and return the results in a form that the doctor can use. The doctor then analyzes the results and combines them to complete the analysis, possibly by repeating sub-activities that were not successful or by invoking new sub-activities.

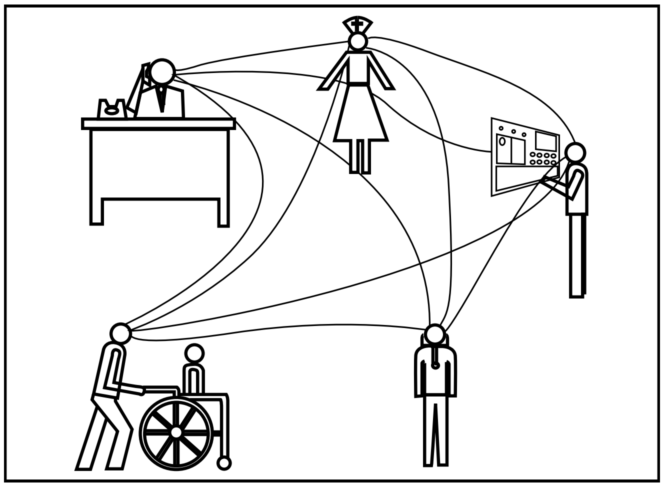

Figure 1. Cooperating agents in medical care.

Such cooperative work requires considerable intelligent interaction among the agents using knowledge of who does what, what information is required, the form in which it is required, scheduling requirements or coordination of tasks, how to locate agents, how to request that sub-activities be done, etc. The cost and quality of products of most human organizations depend on the effectiveness of such cooperation. In hospitals, the quality and cost of health care depend on effectiveness and speed of cooperation. Aspects of the cooperation can be seen as effective parts of the work being done (e.g., doctor's interaction to solve life critical problems), while others may be seen as counterproductive (e.g., converting patient chart information into multiple computer formats for automated analysis steps). The cost and complexity of interactions in a hospital argue for their optimization. What cooperation aspects are effective and should be encouraged, and which should be diminished?

Intuitively, it seems that the distributed computing vision could meet many requirements of cooperative work. The cost of an activity could be reduced by a computing infrastructure that makes appropriate interactions transparent to the agents. Computers could contribute to more productive (e.g., effective and efficient) work by intelligently supporting cooperation. In the next section, I examine forms of intelligence and cooperation that computers might support. I limit my scope to the cooperative work that might be supported by cooperating ISs and the resulting requirements on the computing infrastructure, or systems technology.

1.1.2. Intelligent and Cooperative Information Systems

Intelligent and Cooperative Information Systems (ICISs) are seen as the next generation of IS, 5-10 years in the future [BROD92a]. ICISs are collections of ISs that exhibit forms of cooperation and intelligence. Cooperation is supported by interoperability (the ability to interact effectively to achieve shared goals, e.g., a joint activity). Intelligent refers, in part, to the ability to do this efficiently (i.e., have the system find, acquire, and orchestrate resources in some optimal fashion) and transparently (with the least human effort). The goal is that any computing resource (e.g., data, information, knowledge, function) should be able to transparently and efficiently utilize any other. Although some features of such systems are agreed upon, no one knows the exact nature of these systems. This sub-section illustrates and suggests some initial ideas for ICIS functionality.

Most organizations have developed many application-specific but independent ISs and other computing resources. They soon find that almost all ISs must interact with other ISs or resources, just as the people in their organizations need to interact. Such organizations have vast investments in valuable resources that cannot be used together without great cost. For example, valuable data is bound to applications and is not available to others. There is a growing need for vast numbers of disjoint information/computing resources to be used cooperatively, efficiently, transparently, and easily by human users (e.g., clerks, scientists, engineers, managers). Consider, for example, the different ISs that must interact to support the functions of a hospital (Figure 2). To produce a patient bill, the billing system must obtain information from many hospital ISs (e.g., nursing records, doctors' bills, pharmacy, radiology, lab, ward, food services).

Let's call such an effective combination of systems, a Health Care ICIS. The Health Care ICIS requires access between multiple, heterogeneous, distributed ISs that were independently designed to be used in isolation.

I consider two or more ISs that execute joint tasks to form a larger IS, called a *cooperative IS*. I call an individual IS within a cooperative IS a *component IS*. With various forms of transparency, a cooperative IS can act as, and be considered as, a single IS (e.g., the hospital billing system accesses of multiple ISs should be transparent to the user). A common requirement for component ISs is to maintain autonomy while cooperating within the cooperative IS.

Intelligent features could be added to a cooperative IS. These features require of technology, or provide users with, more *intelligence* than do conventional ISs. Intelligence has a potential role in user interaction between the user and the component ISs to enhance the quality of interaction. Examples of such features include presenting an integrated view of the multiple ISs; explanation; intentional queries; and presenting functionality through graphic, visual, linguistic or other support (e.g., use of icons, templates, graph representations).

Intelligence also plays a role in enhancing IS functionality. Examples include the following:

- Enhanced decision making or reasoning capabilities (e.g., incorporate hospital rules into the Health Care ICIS).
- (Re) Active (e.g., when a new patient is registered, a transaction is triggered that checks the availability of rooms in wards and orders needed supplies).
- Nondeterminism (e.g., give me any one of the possible teams that has two doctors from cardiology, an anesthetist, and three nurses who are not already booked).
- Nondeductive forms of inference (e.g., induction such as learning rules or constraints from databases, reorganizing a schema based on current extensions of different classes, redistributing information based on access patterns; case-based reasoning, where information is structured according to *cases* and new situations are dealt with by finding similar ones in the information).
- Maintaining integrity constraints.
- Introspection: reasoning about meta-knowledge (e.g., a Health Care ICIS component reasoning about what it can and cannot do in the face of a request).

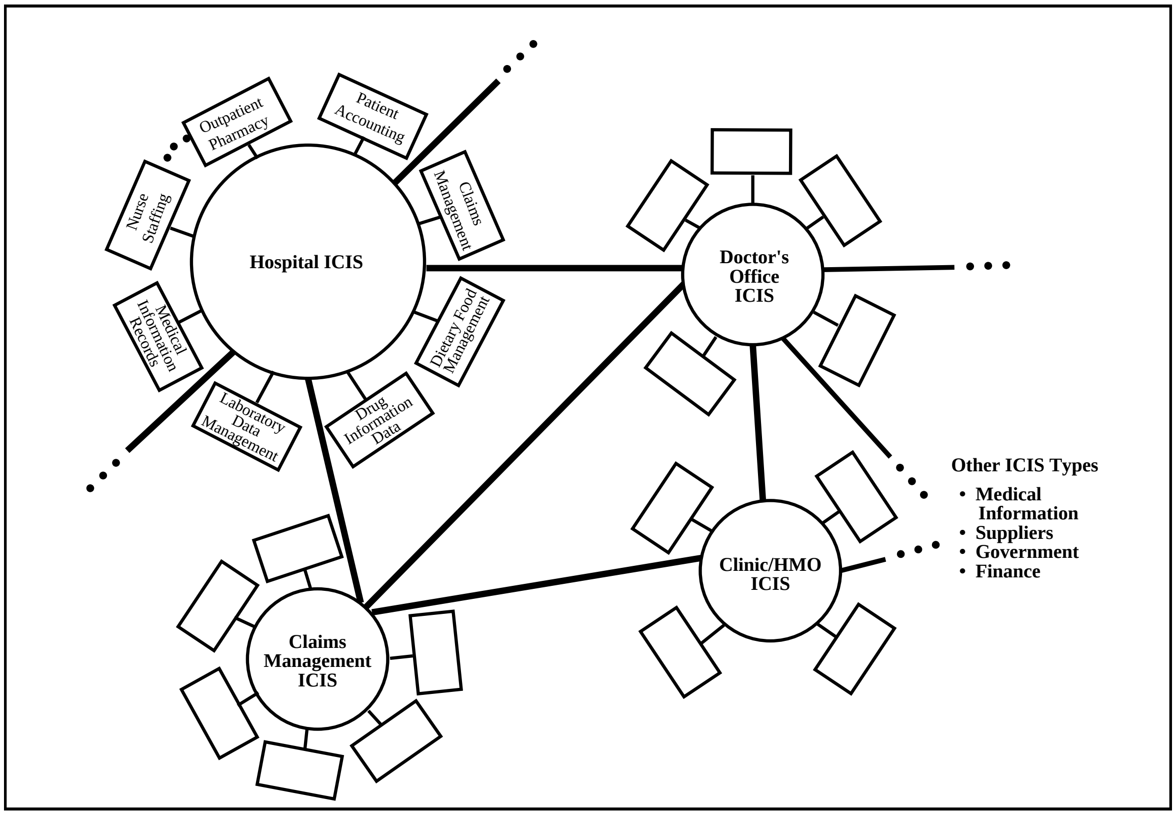

Figure 2. Health care ICIS.

1.1.3. The Global Computer

In a separate universe far away, or so it seems, a vision for the next generation telecommunications technology is taking shape. It intends to permit any information to be communicated anywhere, at any time, in any form, to any agent (human or machine). The key technologies include ubiquitous, broadband, lightning fast intelligent networks enabling universal information access to support information malls, multimedia communications, and business enterprise integration. This will require all-digital, broadband transport and switching; distributed intelligence and high-speed signaling; geographic independence for users and services; interoperability of diverse networks and equipment; transparency via common look and feel interfaces; etc. Sound familiar? Just as computing technology [e.g., database management system (DBMS)] is being extended from computing with text and integer data to computing with all types of objects, telecommunications technology is being extended from communicating voice to communicating all types of information.

The telecommunications vision does not consider only agents in hospitals. A significant difference with the computing vision is the world-wide scale of telecommunications. Figure 3 illustrates agents interacting with agents across Europe and North America. Current telecommunications advances involving cellular communications and satellites will soon permit point to point communication anywhere in the world, with or without old-fashioned wires or new-fangled fiber optics.

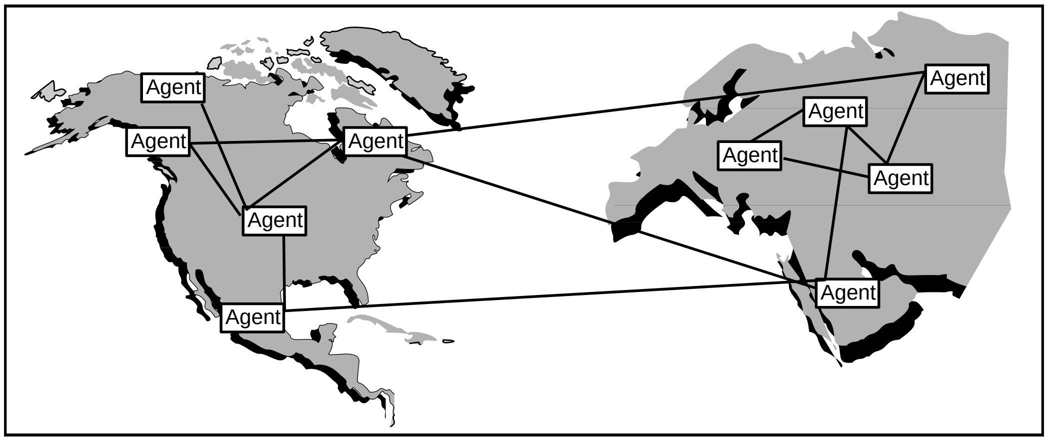

Figure 3. Agent interaction in telecommunications.

There are striking similarities between the computing and telecommunications visions [WILL91]. The motivations are very similar, in terms of both applications (i.e., business) and technology. They both rely on similar technological advances. Indeed, the telecommunications vision, underlying Bellcore's Intelligent Network Architecture (INA) program is stated simply and compellingly as the addition of a distributed computing environment (DCE) or distributed processing environment (DPE) to the public network (i.e., the world-wide telephone system).

The vision of world-wide computing results from the combination of the computing and telecommunications visions. In this vision, the device (e.g., telephone, computer, FAX, television, answering machine) on your desk or in your pocket can be connected to the world-wide public telecommunications/computing network. It has transparent access to all allowable resources in the network. Your computer just became a global computer. You can transparently participate in ICISs world-wide, as illustrated in Figure 4. Figure 4 illustrates Motorola’s Iridium project, which will place over 75 satellites in earth orbit to provide ubiquitous telecommunications. In computing terms, the scale is unimaginable, at least for me. There is currently approximately one device per citizen (i.e., 10^8) on the public network in the United States. How many devices would there be in the global computer?

The telecommunications vision critically depends on computing and *vice versa*. The implications of merging telecommunications and computing are profound. An obvious change will be a lack of distinction between computing and telecommunications technologies and businesses. The world-wide public telephone system becomes a universally accessible DCE/DPE. In broad terms, these visions are widely agreed to. It is not news. Although [KAPL92] predicts that the revolution is at hand,[2] it is taking a long time to realize the visions. New technology facilitates revolutions; it seldom brings them about. These visions are currently solutions looking for problems. For example, two major telecommunication/computing experiments in the United States have no compelling applications to drive them. The most advanced technology is being used to support computing and communicating hospital imagery and visualization of scientific data. These are not the killer applications that are in such great demand that they alone will force the realization of the visions. Other problems concern unreliability, privacy, security, fraud, and hype.

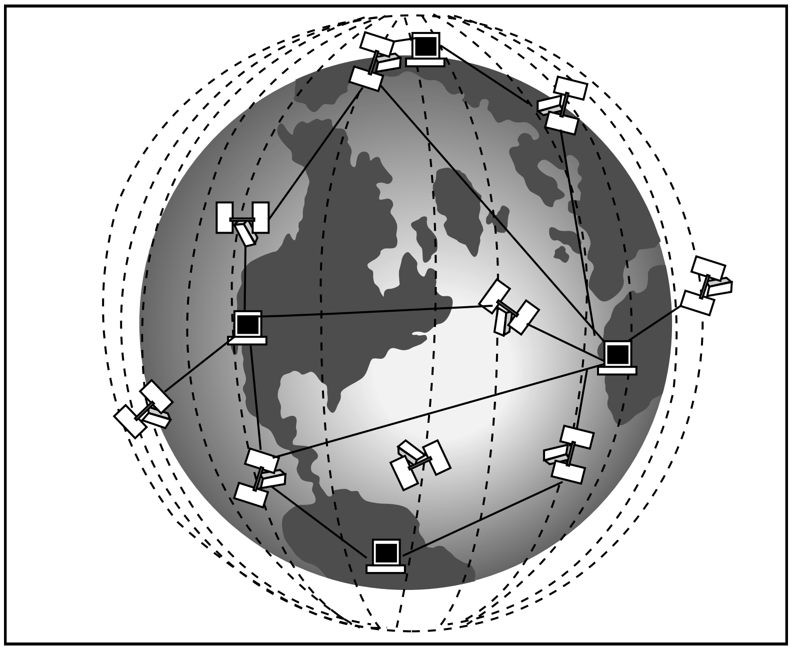

Figure 4. World-wide computing.

The remainder of the paper addresses technical ideas and challenges in the realization of the distributed computing vision, possibly on the scale of world-wide computing. In addition to basic research challenges, there are the critical challenges of evolving the current technology base, including existing or legacy ISs, towards the vision. I return at the end of the paper to the lack of killer applications.

[2]"As a result of rapid technological developments in the computer software and hardware, consumer electronics, and telecommunication industries, we believe a true revolution in the delivery of entertainment, information, transactional, and telecommunication services may be at hand."

1.2. The Technology

The motivations for the telecommunication and distributed computing visions are similar. The primary motivation is for users to gain control of their technology/networks (e.g., distribution and deployment of their data and applications). In the past, hardware, and to some extent software, vendors had control (e.g., use any system or computer you like as long as it is Blue). A current goal is vendor independence as expressed in the ill-defined phrase *Open Systems*. Another motivation is the dramatic cost reductions possible in moving computing from costly mainframes to workstations/minicomputers. These strategic motivations relate to key technical motivations such as interoperability and reuse of interchangeable components. The potential magic of these trends towards open, distributed computing/telecommunications is that they potentially support what appears to be more natural modes of working/playing for humans and organizations (i.e., cooperative work). The magic is still potential due to the lack of killer applications. This sub-section describes a pervasive vision of next generation computing architectures, the generalization to this environment of database concepts, and consequent research challenges. This sets the stage for the subsequent section, which poses the challenges of legacy ISs in the face of this vision.

1.2.1. Next Generation Computing Architecture and Object Space

The agent-oriented, cooperative work described above will be supported by a global object space, as illustrated in Figure 5. Cooperating agents (e.g., resources such as humans, computers, information bases) will interface to the global object space via a human or systems interface, depending on whether the agent is a human or a system. Each agent may request or provide resources to the global object space (e.g., may be a client or a server). As a server, the agent may provide all or some subset of its capabilities. The interface layer will provide transparent access to the object space by mediating between the agent's languages, formats, etc., and those of the object space. By the mid 1990's, the notion of a global object space will become an architectural paradigm which will provide the basis of next generation computing environments. These environments will provide practical interoperability with limited transparency, efficiency, intelligence, and distribution.

The object space can be considered a logical extension of a database. A database consists of data elements. Using the object-oriented approach, data and operations are encapsulated so that all computing resources (e.g., data, information, programs, knowledge) can be seen as objects. Data management concepts are being extended to manage objects. Hence, the global object space provides a challenging opportunity for extending database technology to global object management. Given the world-wide computing vision, the scale of the challenges is enormous. The opportunities for distributed computing are compelling.

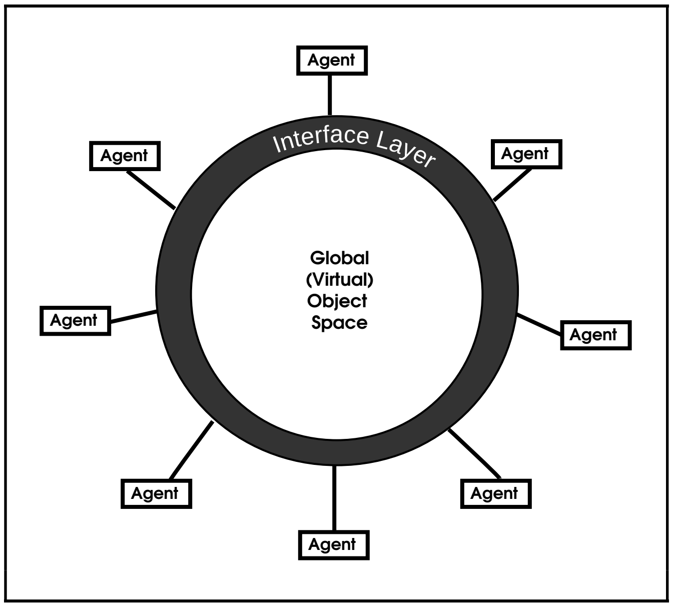

Figure 5. The global object space.

The global object manager could potentially offer access to any subset of the union of the resources provided by the agents (e.g., agent specific object spaces or object views). The object space may be largely virtual since it is an implementation detail as to how the objects (i.e., agent-provided resources) are managed, stored, and accessed. A global object manager will support efficient and intelligent (e.g., transparent) interoperability between all objects. Hence, resources can be combined in unanticipated ways to meet new application requirements and opportunities. The object space will be based on the object-oriented approach and will be supported by a generic technology which will provide tool kits for combining computing components in arbitrary ways. In this context, the notion of a single DBMS as a separate component serving a wide range of applications will cease to exist. Instead, DBMSs become cooperating agents. Database interoperability will be supported by the global object space manager just as it supports interoperability between all agents.

Object space management technology is only one of many systems technologies necessary to support the vision. The remarkably successful database notion of extracting as much data management functionality from applications as possible and providing it in a generic DBMS (Figure 6) has been extrapolated to many computing services to arrive at a distributed computing architecture, illustrated in Figure 6. The resulting architecture separates four systems functions:

- User interfaces
- Applications (the minimum code necessary for the application semantics)
- Shared distributed computing services (e.g., global object space management)
- A global object space

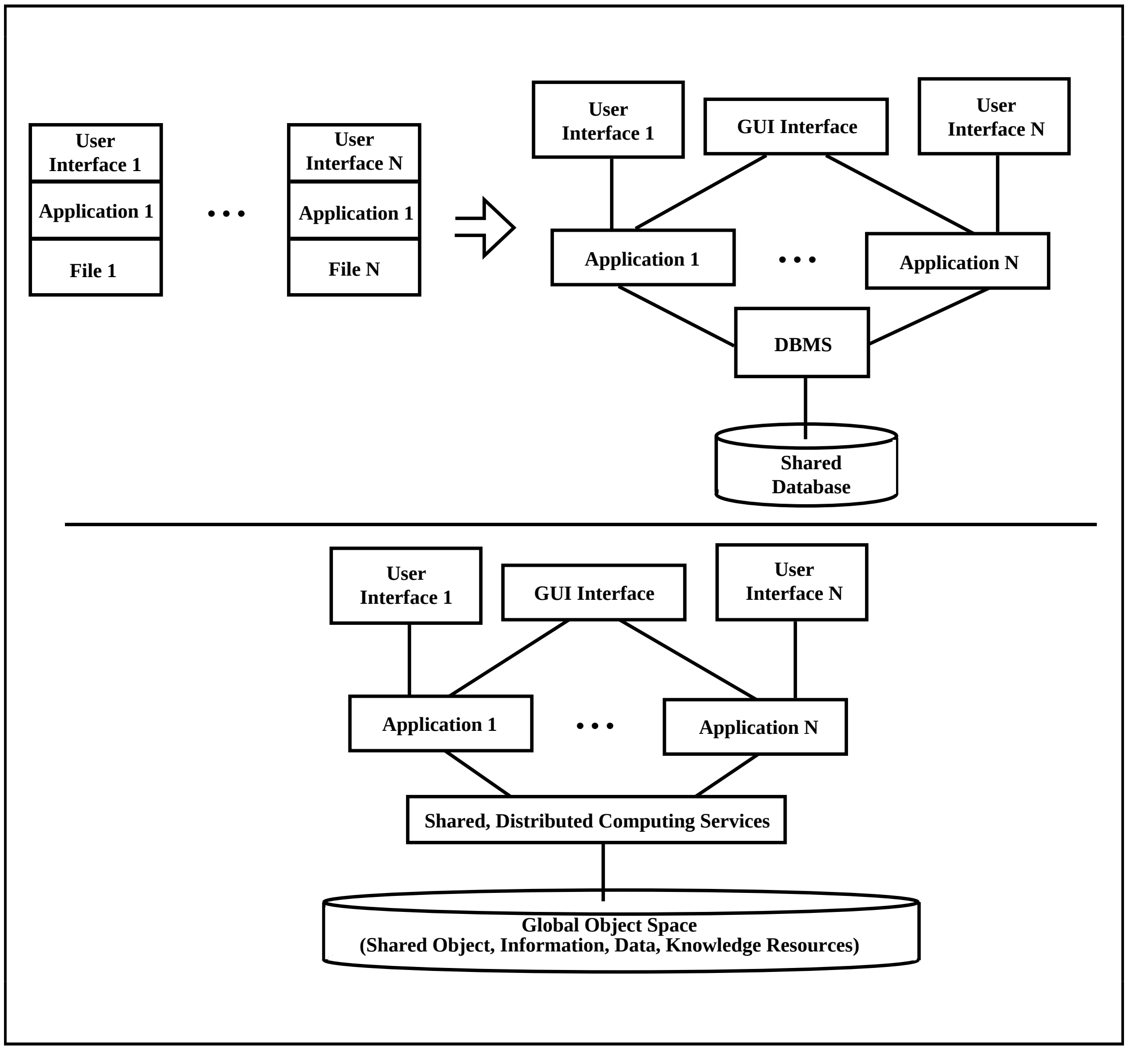

Figure 6. Distributed computing architecture.

The most compelling idea of the distributed computing architecture (also referred to as the Enterprise Information Architecture) is the concept of shared, distributed computing services. The principle is to provide as many computing services as possible as sharable servers in a distributed computing environment. The services are provided by systems or computing infrastructure technologies such as DBMSs, OSs, user interface management systems, and development environments. This is the home of the global object manager. Collectively, the systems providing infrastructure support are being called **middleware.** Figure 7 illustrates details of the layers of the architecture:

- Human Presentation Interface
- Applications and System Applications Development Environments
- APIs
- Middleware (Distributed Computing Services)
- System Interfaces
- OS, Network, and Hardware and gives examples of the middleware.

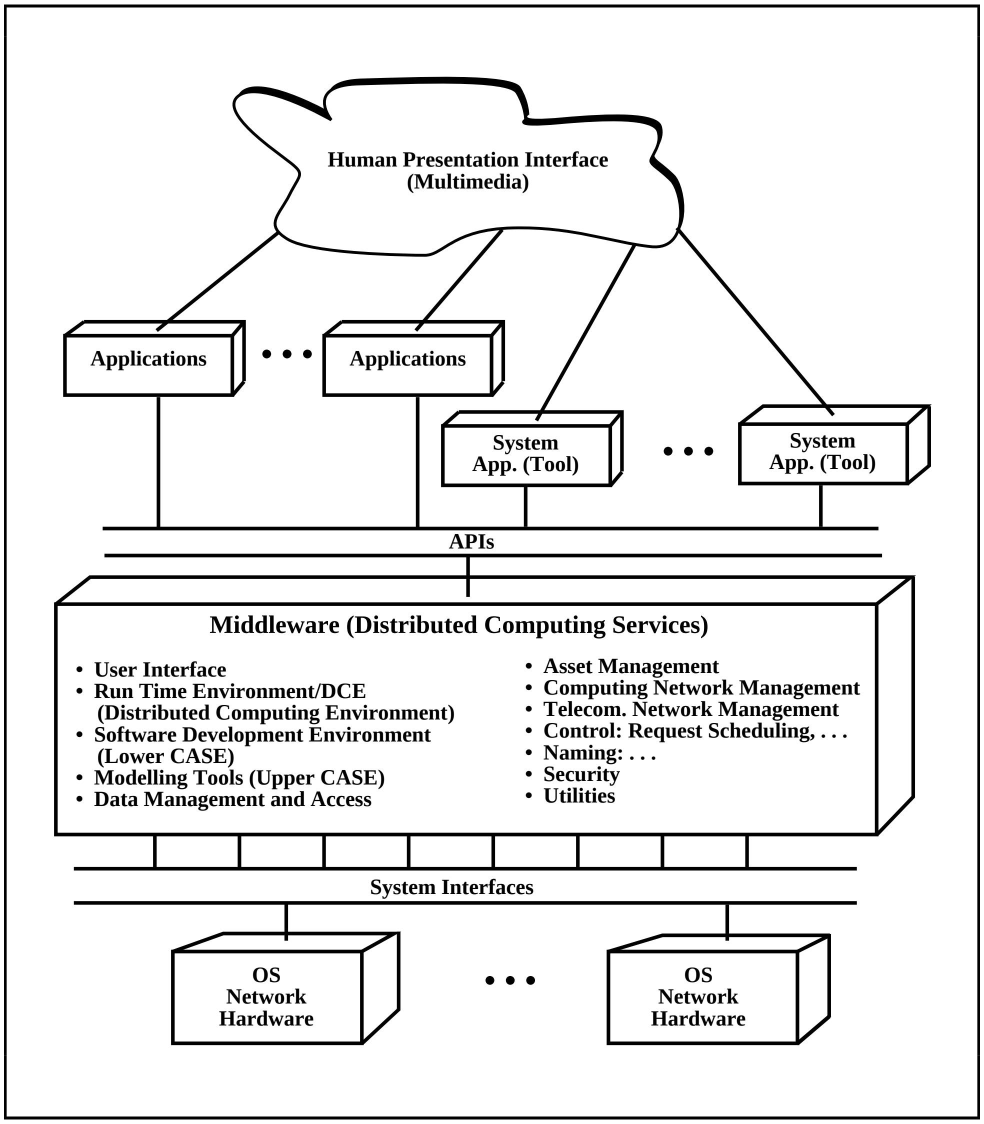
Human Presentation Interface
(Multimedia)
Applications
Applications
System App. (Tool)
System App. (Tool)
APIs
Middleware (Distributed Computing Services)
• User Interface
• Run Time Environment/DCE (Distributed Computing Environment)
• Software Development Environment (Lower CASE)
• Modelling Tools (Upper CASE)
• Data Management and Access
• Asset Management
• Computing Network Management
• Telecom. Network Management
• Control: Request Scheduling, . . .
• Naming: . . .
• Security
• Utilities
System Interfaces
OS Network Hardware
OS Network Hardware

Figure 7. Enterprise information architecture with middleware.

The distributed computing architectures, middleware, and global object management are much more than visions. The major software vendors have all announced their support of the architectural goals, including the following:

- Open systems (i.e., standard interfaces or architectures)
- Client-server computing
- Distributed computing, including distributed DBMSs
- Advanced flexible user interfaces

- Transparent multivendor interoperability
- Transaction processing
- Structured organization of corporate resources/components
- Reliability and maintainability
- Reduced communication costs
- Single-point program maintenance and upgrading
- Open access to low-cost workstation MIPS

Some vendors have announced and released middleware and related products and strategies, including the following:

- Enterprise Network Architecture (ENA) (Gartner Group terminology)
- Network Application Architecture and Network Application Support products (DEC)
- SAA: System Application Architecture (IBM)
- Information Warehouse Architecture (IBM)
- DRDA: Distributed Relational Database Architecture (IBM)
- Advanced Networked Systems Architecture (ANSA) [ARM89]
- SOE: Systems/Standard Operating Environment (pervasive)
- DCE: Distributed Computing Environment (Open Systems Foundation)
- DME: Distributed Management Environment (Open Systems Foundation)
- GAIA: GUI-based Application Interoperability Architecture (Open Systems Foundation)
- OSI Network Management Forum
- Integration Architecture for the Enterprise (Apple): Data Access Language, MacWorkStation, MacAPPC (LU 6.2)
- Open Application Development Architecture and Open Enterprise Networking (Tandem)
- Bull's: Distributed Computing Model (DCM)

There is considerable activity towards achieving interoperability between arbitrary computing services via object-orientation and distributed computing technologies such as distributed databases. There are several research, product, and standards development projects exploring different aspects of global object management. The projects originate in disjoint technologies, including databases (e.g., GTE's Distributed Object Management [MANO92a], Open Systems Liaison's Object Mapper, Stanford's Mediator-based SoD and KSYS, the University of Florida's Federated Information Bases), programming languages (e.g., MIT's Argus), operating systems (e.g., Open Systems Foundation's OSF/1, ANSI’s and ISO’s Open Distributed Processing, APM Ltd's ANSA [ARM89]), communications (e.g., Open System Liaison's Software Bus), software engineering (e.g., Norway's Eureka Software Factory), and in new technologies that combine features of several technologies (e.g., Yale University's LINDA [CARR89]). The use of the object abstraction in integrating heterogeneous and autonomous components is also a characteristic of recent developments in personal computer application integration software, such as Hewlett-Packard's NewWave, Object Linking and Embedding (OLE) in Microsoft® Windows™, and inter-application communication (IAC) facilities in the latest Apple Macintosh® operating system.

Perhaps the most notable activity in global object management as a basis for interoperability is that of the Object Management Group (OMG). OMG is an industry consortium of 270 of the major software and hardware vendors around the world. Its goal is the development of a framework of specifications to maximize the portability, reusability, and interoperability of commercial object-oriented software. Within two years, it has proposed a guideline for a Common Object Request Broker Architecture (CORBA) [OMG91a] which provides the core functionality for interoperability to be provided by the global object manager. It facilitates the exchange of messages between arbitrary objects in a distributed, heterogeneous computing environment. The broader goal of CORBA is true application integration by providing a common approach to persistence, transactions, security, naming, trading, and object services. The ORB serves as a communications substrate (like RPC) which is intended to support object implementations as diverse as separate processes, code in the same address space, object databases, etc. CORBA’s interoperability can be provided to any software that is capable of sending and receiving messages. Hence, it is a basis for interoperability between arbitrary computing systems, whether object-oriented or not. As of mid 1992, over 15 major software vendors have announced CORBA-compliant ORBs. Over 50 OMG members, including IBM, have internally adopted the CORBA specification.

Although object-orientation is in its infancy and is not deployed in practice to any significant degree, it plays a critical role in all of the visions described above. There is a pervasive industry commitment to object-orientation. This is demonstrated by the following strategies and products announced by major vendors:

- IBM's System View (OO is seen as critical to manage large-scale systems)
- IBM-Apple agreement based in part on OO technology
- Apple uses OO in many of its current products
- TINA's commitment to OO (a Telecommunications pre-standards body)
- Multivendor networks based on "Managed Objects"
- Microsoft's Object Linking and Embedding (OLE)
- Microsoft’s CAIRO (approach to distributed objects)
- Hewlett-Packard’s Distributed Object Computing Program (DOCP)
- Sun’s Distributed Objects Everywhere (DOE)
- IBM-Metaphor's Patriot Partner's Constellation Project
- DEC's Trellis OO Application Development Environment
- OODBMSs or DBMS support for OO: Ingres, Oracle, Objectivity/DB (DEC-Objectivity), Ontos (IBM-Ontos), and 13 other OODBMS products
- Object Management Group’s (OMG) CORBA, core object model, object services, and common facilities; fifteen companies have announced CORBA-compliant products

Another indication of the potential significance of and high expectations for object-orientation is the number of object-orients standards and consortia, including the following:

- ANSI X3H2-SQL (object-oriented support in SQL3)
- ANSI X3H4 Information Resource Dictionary Systems
- ANSI X3H6 CASE Integration Service
- ANSI X3H7 Object Information Management [FONG91]
- ANSI X3J4 Object COBOL
- ANSI X3J9 Pascal
- ANSI X3J16 C++
- ANSI X3T3-ODP (Open Distributed Processing)
- ANSI X3T1M1.5 (related to ODP)
- ANSI X3T5-TP (Transaction Processing)
- ANSI X3T5-OSI (Open Systems Interconnection)
- ISO’s IRDS Information Resource Directory Systems
- OSI/Network Management Forum
- JTC1 SC21/WG4 (Management of Information Services) CCITT's "Managed Objects"
- OSF's Distributed Management Environment
- CCITT’s TINA Telecommunications Information Networking Architecture [WILL91]
- Object Management Group (OMG)
- OSF Open Systems Foundation
- X/Open
- PCTE: Portable Common Tools Environment

[Fong91] lists 32 related computing standards efforts in DBMS, transaction processing, object communications and distribution, data interchange, domain-specific data representations, repositories, programming languages, and frameworks and consortia. The TINA 90 and TINA 91 proceedings (see e.g., [WIL91]) lists a similar number of standards and framework efforts in telecommunications and computing.

The existence of so many object-oriented standards efforts poses potential problems of inconsistent object-oriented standards, thus hindering interoperability, a goal of object-orientation. The ANSI standards body X3H7 was established, in part, to address these problem in the area of object information management. X3H7 has been chartered to make more consistent the object facilities used in various standards (including those of both official standards bodies and industry consortia), where such consistency would improve their ability to interoperate. One of its initial goals is to produce a reference model which could be the basis of later standards. The group is currently discussing whether it should be producing a new object model (e.g., a *least common denominator* model) [MANO92b] or just facilities for describing object models. This challenge is directly related to data and object model research in databases.

Next generation computing architectures are well on their way from vision to reality. There is a pervasive agreement on many aspects of the vision, including the architectural notion of middleware, the functionality of (global) object management, and the critical role of object-orientation. There is less agreement on the need or means for integrating technologies so as to draw the greatest benefit from each technology, rather then re-inventing the wheel. These trends are led and largely determined by the computing industry and will determine, to a very large degree, computing environments for a long time to come. As described in the next sub-section, basic research challenges must be met to realize the vision. To date, the research community has had little impact on formulating and realizing the vision. This is a major opportunity for the research community to do excellent research and make significant contributions.

1.2.2. From Database to Object Space Management

Global object space management can be seen as a logical extension of database management to the global object space. Database technology contributes such concepts on which to base object models, persistence, sharing, object management and migration, optimization, transactions, recovery, distribution, and heterogeneous database interoperability. However, the global object space poses additional challenges, not only based on the scale of the object space compared with that of a database. Global object management requirements are dramatically different based on the fact that heterogeneous objects are being managed. A critical, new requirement of global object management is support for general-purpose interoperability. This sub-section defines interoperability, outlines an approach to providing interoperability, and lists related basic research challenges.

1.2.3. Next Generation Database Research

ICISs were defined above in terms of the ability of two or more systems to interact to execute tasks jointly. This capability is the intuition underlying interoperability. However, there is no agreement on the functionality implied by the term *interoperability*. Therefore, I provide an initial definition and a discussion of the idea and of related objectives.

Interoperability: Two components (or objects) X and Y can interoperate (are interoperable) if X can send requests for services (or messages) R to Y based on a mutual understanding of R by X and Y, and Y can return responses S to X based on a mutual understanding of S as (respectively) responses to R by X and Y.

This definition addresses the function of interoperability and not aspects of the systems context in which it is provided. For example, two components (or objects) that call each other and that are written in the same language, under the same operating system, and on the same machine illustrate trivial interoperability. Specific systems challenges arise in providing interoperability by a DBMS to applications written using the DBMS. More challenges arise when providing it to applications written over heterogeneous, distributed DBMSs. Even more arise when providing it over arbitrary computer systems that may not support any DBMS functionality.

Interoperability does not require X or Y to provide both *client* and *server* functionality, to be distributed or heterogeneous, or to provide any forms of *transparency* with respect to each other unless such transparency is required to satisfy the above definition. However, given a particular systems context, interoperability may require some or all of these features. Alternatively, interoperability could be improved or its scope increased by advances in technology that increase the ease, efficiency, reliability, and security with which components interoperate. Some of these advances can be expressed in terms of various forms of transparency in which differences are hidden and a single, homogeneous view is provided. The following forms of transparency are generalized from those proposed for distributed databases.

- Resource (e.g., seeing one system or resource provider versus needing to know the individual system(s) providing the service or information).
- Language (i.e., using one language and therefore not needing to know the language of the resource).
- Distribution (i.e., not needing to know the location(s) of the resource(s) or how to transfer to and from them).
- Logical/schema (i.e., having the appearance of one meta-information base describing such things as how features are modelled in individual resources).
- Transaction (i.e., ability to run one, apparently local, transaction over multiple resources that appear as one resource); transactions act the same (atomic, commit, abort) at one, two, or more sites.
- Copy/replication (e.g., not needing to know that resources are replicated).
- Performance (i.e., tasks execute efficiently independently of the location of invocation or of participating resources).
- Data representation (i.e., not needing to know how information is stored).
- Fragmentation (i.e., not needing to know that information is fragmented).
- Location (i.e., programs access a single logical database, not needing to know the location of the actual data).
- Advanced application development (i.e., supports the creation of *business rules* to be applied to all distributed processing);
- Local autonomy for participant ICISs (i.e., individual components can access and rely on all resources; local operations are purely local, all components are treated equally).
- Network/communication (i.e., communication protocols are transparent).
- Hardware (i.e., execute components ignoring hardware platform).
- Operating system (i.e., run components ignoring the OS that they will run under).

These and other forms of transparency could be described from a system level perspective, e.g., consistency/integrity of copies of object/data/knowledge; augmenting systems with features to ensure system-wide integrity (e.g., backup and recovery). Interoperability involves far more than transparency. It involves issues that arise when two languages or systems must interact, including those involved with type systems, transaction systems, communication protocols, optimization, and systems architecture. These requirements pose basic research challenges to otherwise manageable database technology challenges.

Interoperability could be characterized as a form of *systems level intelligence* that enhances the cooperation between ICIS components. Consider the intelligence required to provide services, find resources, cooperate and carry out complex functions across component ISs without the user or component IS needing to know precisely what resources are available, how to acquire them, or how they will be orchestrated to achieve the desired result.

Let's consider an approach to providing global object management. This approach is that of GTE's Distributed Object Management project [MANO92a] and of the OMG [OMG91a]. The scale and distribution of the object space leads to the distribution of global object management functionality into a collection of distributed object managers (DOMs). Figure 8 illustrates, in the Health Care ICIS, that medical agents, possibly assisted or simulated by ICIS components, are interconnected indirectly through DOMs.

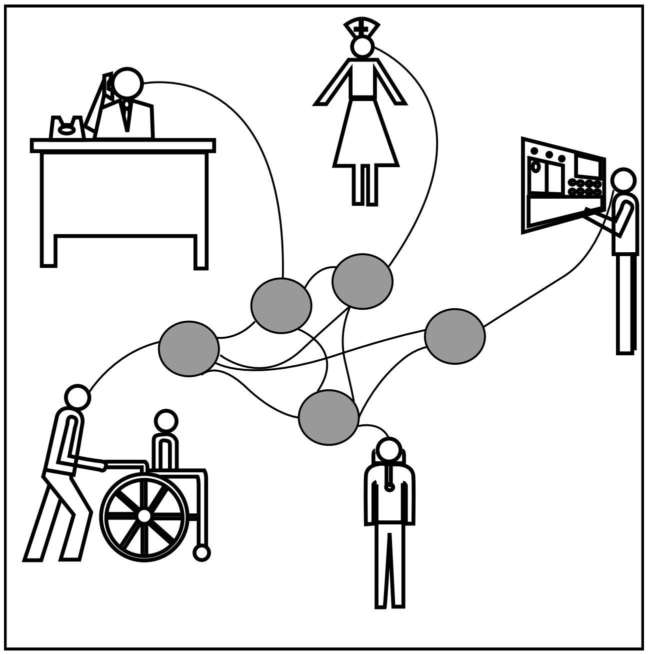

Figure 8. Health care ICIS interoperability via DOMs.

In the DOM approach, each resource has a client interface to the DOM, as illustrated in Figure 9. All DOMs have one common object model. Interfaces between resources and DOMs allow resources to be accessed as objects in the common object model. For clients, interfaces allow access to objects, and translate requests and results between the local type system and the common object model.

Consider as an example interaction a request by an object-oriented DBMS to copy text on what appears to it to be a document object (Figure 10). The request is routed by DOM of System 1 to the DOM of System 2. The interface to the nonobject-oriented word processor allows its DOM to treat files as document objects. The System 2 DOM invokes the word processor, causes loading of the references text file, and invokes the requested operation via the interface.

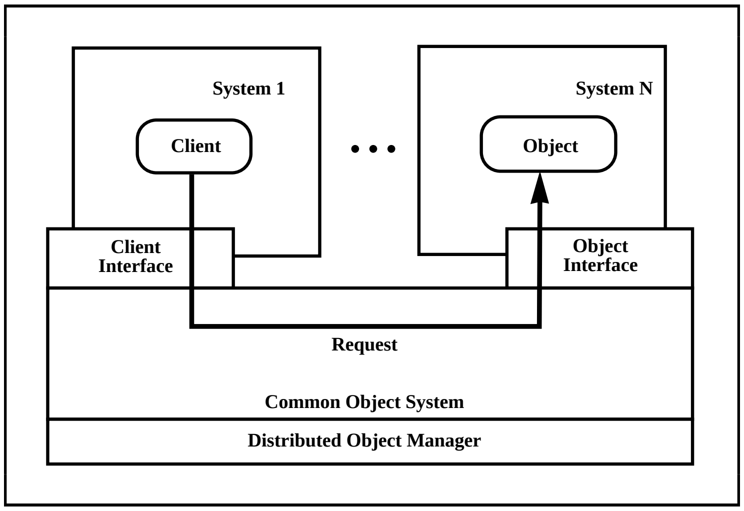

Figure 9. Role of a DOM.

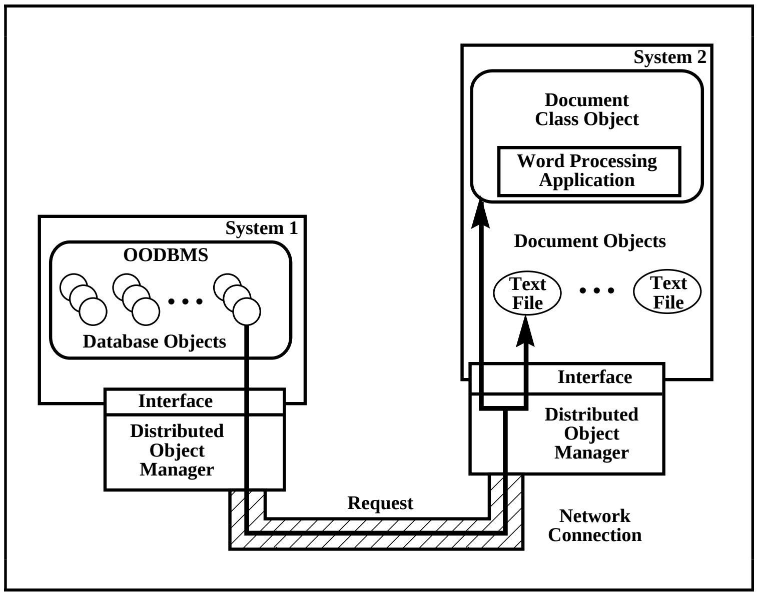

Figure 10. OODBMS client request of a word processor.

Global object management involves all database research issues except that in the context of the global object space, the challenges are greater. Database solutions may not extrapolate to adequately address global object management requirements. The following is a list of global object management research challenges that appear significantly more difficult than the database counterpart.

- Interoperable object model: It must support all requirements of the multiparadigm, distributed computing environment, not just object manipulation [MANO90]. It must provide a basis for mapping between arbitrary computing systems, not just DBMSs [OMG91b]. The proliferation of object model and object model standards leads to a requirement for a *least common denominator* or *RISC* object model [MANO92b]. A challenge is to develop an object algebra which is for objects what the relational algebra is for relations. This would provide a basis for a theory of objects and for object DBMSs as relational algebra does for relational DBMSs.

- Long-lived, distributed applications (i.e., the ICIS counterpart to database transactions): An ICIS operation potentially involves operations over many distinct component ISs. Writing transactions corresponds to manipulating entire applications using operations like: start, abort, cancel; suspend, resume, backtrack; migrate computation; and show computational state/history. This requires new abstractions for *programming in the large*, new control mechanisms, new notions of correctness, and compensation. In short, new transaction models (e.g., open, nested, complex, multisystem operations) are needed [GRAY93] [ELMA92]. Unlike the conventional transaction model which is orthogonal to the relational data model, the DOM transaction model will have to be integrated with the DOM object model.

  Multisystem transactions and workflows are examples of *programming in the large*. They involve programming in which entire ISs are encapsulated and provide functions or services in support of a higher level operation (i.e., the transaction).

- Query optimization: Optimization of queries over the global object space involves optimizing operations over multiple database and nondatabase systems, vastly expanding the scope for optimization by potentially applying database optimization technology to arbitrary computing tasks.

- Global naming: Objects should be uniquely identifiable throughout their life regardless of their system of origin and current location [ZATT92]. Objects or references to them may be embedded in a large number of heterogeneous contexts throughout the global object space. Each context (e.g., IS) will likely have its own unique naming scheme and will require considerable autonomy. A challenge is to provide a mechanism to register a context, including some of its objects, in the global object space that would accommodate many, co-existing, heterogeneous naming schemes within a common (structured) global naming scheme. Another challenge is to develop a migration path that would allow legacy ISs to participate in the naming scheme but permit considerable autonomy. How many objects do you think this might involve?

  Naming, as discussed above, involves agreements on what I will call identical concepts. This includes not only identity of values (e.g., Fred's age here is identical to Fred's age there), but also identity of operations (i.e., that the results of executions in the identified context are identical as agreed to by any object that might then reference the result). For example, I want to establish that the *meaning* of PAY(EMPLOYEE) is identical in two contexts (e.g., Fred would get the same pay in two companies if the contexts were identical) except for the differences introduced by the contexts.

- (Object) Views: I assume that complete semantic integration of all ISs is infeasible. Hence, mapping between ISs requires the ability to sub-set the respective ISs so as to present the appropriate information (objects) to the other ISs. This amounts to an object-view (analogous to a relational view). Views in relational databases still pose major challenges, such as view updates. In object-oriented systems, the problems are more difficult. [HEIL90] [SCHI89]

- Active objects for active ICISs: Cooperative human interaction to achieve jointly agreed upon tasks requires response to conditions, actions, and events in the working environment. Analogously, ICISs must be able to respond to changing conditions, actions, and events. Techniques to facilitate these requirements include rules (If C, do A), events (When E then If C then Do A), triggers, and alerters. These techniques make systems active in that they now react to changes. These techniques create complex temporal and conditional relationships between objects.

- Modelling: Conceptual modelling has not enjoyed the successes of other database technologies. ICIS complexity, scale, and heterogeneity make the problems dramatically worse. Due to the early stage of development and experience, there are currently no effective, proven concepts, tools, or techniques for object-oriented systems design of the scale being considered here. How do you visualize or conceive of all the objects in a large IS? How do you design their interactions and ensure that all possible combinations of method invocations are meaningful and correct? How do you control the invocation of methods to ensure properties such as integrity, correctness, and recoverability. We have almost no means of *designing in the large* (e.g., design at a level above entire, encapsulated ISs to define multi-IS workflows or transactions). Whereas the semantic aspects of interoperable ISs are not different than conventional ISs (see below), new approaches to modelling are arising from some new aspects of distributed computing including notions of cooperation and agent-oriented computing, object-orientation, advanced transaction models, workflows, active objects (e.g., event-based and rule-based computing), and the necessarily higher level view of ISs.

- Gateways: Gateways will be major components in future IS architectures for both new and legacy ISs. Interoperability involves the interfacing of computing systems. Whenever two systems are required to interact, some interface is required. If the interactions involve anything more than simple message passing, it is often necessary to develop a software interface component. Such a component is called a gateway (i.e., between the two systems). Gateways are an integral part of distributed DBMS architectures for heterogeneous DBMSs. Database gateways are complex, costly, and generally *ad hoc* (i.e., built solely for the two systems being interfaced). They also play a major role in encapsulating legacy ISs when migrating legacy ISs into new software and hardware platforms [BROD92b]. In this regard, they are sometimes referred to as surround technology. They will become fundamental elements in IS and ICIS architectures (Figures 5, 7, 9, and 10). Gateways have stringent requirements. For example, those between mission critical systems must be very reliable, robust, and efficient. They cannot impose a significant performance penalty.

  Requirements for systems interaction (misleadingly called systems integration) and distributed computing (e.g., client-server) are growing dramatically. Gateways are a frequently proposed solution. Indeed, some proposals for next generation computing [OMG91a, OMG91b, MANO92, BROD92b, BROD92a] are based on gateways, which are related to stubs generated by some compilers; ORBs, adapters, and IDL specifications [OMG]; ANSA traders; APIs, and LAIs [MANO92]. Research is required to develop generic and efficient gateway technology for the automatic generation of gateways between arbitrary computer systems. There are an increasing number of specific (i.e., system to system) and minimal gateway products. General purpose tools are required for constructing gateways, just as DBMSs are generic tools for constructing data-intensive applications.

- Semantic aspects of interoperability:[3] Intuitively, semantic considerations for interaction concern establishing that you and I mean the same thing by the messages we exchange. In the definition given above for interoperability, this concerns the mutual agreement of components X and Y on the messages R and S that they exchange. Interoperability, as defined above, is based on the object-oriented notion that objects are defined in terms of their interfaces which encapsulate both their values and methods. In the definition, component, or object, Y has an interface that includes the methods that can respond to R and that X must understand in order to interoperate appropriately with Y. Similarly, X has an interface that includes the methods that can respond to S and that Y must understand in order to interoperate appropriately with X. That is, the relationships between X and Y are defined by R and S. Whereas relationships in semantic data models were defined in terms of structure, predicates, and functions, relationships in object-oriented models are defined in terms of the messages that the objects can

[3]Semantic interoperability presumably refers to the semantic aspects of interoperability. However, interoperability requires that semantic aspects be addressed. Is there another kind of interoperability in which semantics should not be addressed? What is non-semantic interoperability (c.f., There are other kinds of databases then relational? There are non-relational databases). Hence, I will use that phrase *semantic aspects of interoperability*, and not the term semantic interoperability.

meaningfully exchange (i.e., that they mutually understand). Hence, interoperability, as defined above, includes the semantic relationships between objects.

In general, semantic aspects of interoperability between two interoperable objects involves establishing, representing, maintaining, and possibly evolving through time, the relationships between those objects. Operationally in object-oriented environments, relationships are implemented through the messages they exchange and are manifest in the results of the objects' responses to the messages.

As discussed in detail above, the definition does not address issues such as whether X or Y are in the same or in different ISs. When X and Y are in different ISs, there are more issues to address than when they are in the same IS. What can become more difficult is establishing, representing, implementing, and maintaining relationships. However, the semantic relationships should be the same in either case. Indeed, from the semantic point of view, all objects that are registered in the global object space can be considered to form one object space independently of ISs boundaries. Hence, all of the concepts, tools, and techniques for dealing with semantic aspects of single or distributed databases or ISs (e.g., those of conceptual modelling [BROD84]) apply to the semantics of interoperable ISs (i.e., ICISs). Unfortunately, to date, conceptual modelling has had little impact on ISs [BROD92a]. It has not enjoyed the clear progress, over the past 25 years, shared by many systems areas, including operation systems, databases, and programming languages. Problems of semantics in ISs are inherently hard and are fundamentally important. No matter how powerful our computing technology, semantic aspects will remain in the area of open problems.

This observation does not simplify the semantic problems of interoperable ISs. Rather, it demonstrates that the problems are at least as hard as individual ISs. No new semantic problems have been added except scale (i.e., a vastly larger object space), and various forms of heterogeneity (e.g., such as those listed above in the discussion of transparency). In addition, the environment is object-oriented, the components may be in separate ISs, and the ISs may be built without a database or schema, in the conventional sense.

I conclude this discussion with one semantic problem in interoperable ISs. It and others are illustrated in the next section. The scale of the global object space for interoperable ISs makes the notion of a global schema infeasible. Indeed, it may be best to have no global anything except a global naming scheme. Interoperability between two objects requires, at a minimum, that the objects be able to reference each other and that when they mutually refer to each other or to a third object using a particular name, they mean the same object. Global naming establishes identity only. When you have identity between two objects, their semantic relationship is trivial. When you cannot establish identity between two objects that you understand to be identical, you have semantic problems. Solutions may involve modifying one or both objects or their contexts until the desired mapping (e.g., identity) can be established. Related problems include schema mapping, and proving programs equivalent. While DOMs or ORBs do not solve semantic integration problem, their type systems and mapping mechanisms (e.g., client interfaces, adapters, stubs) provide powerful tools for mapping between systems. First-order logic also provides powerful means for such mappings.

Requirements for global object management will come, in part, from the applications that require such services, such as ICISs. The following is a list of basic research questions concerning the nature of ICIS [BROD92a]:

- What forms of cooperation are required between components in an ICIS?
- What ideal architecture (components and interfaces) is required to support ICISs?
- What are the modelling/programming paradigm requirements of an ICIS?
- What languages are required in the life-cycle of an ICIS?
- What does it mean for an ICIS or object to be (re)active?
- What forms of intelligent functionality are required for ICISs?
- What are the requirements for the repository or global directory service?
- What forms of interoperability will ICISs require?

- What are the transaction model requirements of an ICIS?
- What are the key optimization challenges in support of ICISs?
- How can core technology support the inevitable evolution of large-scale ICISs.

Due to the key role of object-orientation, it is important to realize the basic research challenges that exist in that domain. Currently there is no underlying theory comparable to the relational model for databases [BEER90]. Hence, there are not general-purpose means for query or transaction optimization. Further, there are inadequate means for dealing with the complexity and scale of object-oriented systems. There are inadequate concepts, techniques, and tools to model large (e.g., 200+) object systems. In terms of programming and execution, how can you describe and prove that the spreading invocation of operations will achieve some specific functionality? How do you optimally schedule the execution of such operations? Do objects schedule themselves or provide other systems functions such as concurrency control and security or is this done outside the object?

We may require new ways of thinking. The conventional idea of defining each specific relationship between an object and all potentially related objects can't be right. Consider how bird and fish *objects* interact in nature. Defining every relationship between 100 fish in a small tank may be doable, especially when you can see them all. Do fish really work like that? What happens in a boundless ocean with unlimited fish? Thousands of fish move beautifully in schools, etc., probably without sending messages between every two fish. Similarly, how do large flocks of birds fly in formation?

2. The Challenge of LEGACY INFORMATION SYSTEMs

Most large organizations are deeply mired in their IS *sins of the past*. Typically, their ISs are very large (e.g., 10^7 lines of code), geriatric (e.g., more than 10 years old), written in COBOL, and use a legacy database service (e.g., IBM's IMS or no DBMS at all). These ISs are mission critical (i.e., essential to the organization's business) and must be operational at all times. These characteristics define *legacy information systems*. Today, legacy ISs pose one of the most serious problems for large organizations. Costs due to legacy IS problems often exceed hundreds of millions of dollars per year. They are not only inordinately expensive to maintain, but also inflexible (i.e., difficult to adapt to changing business needs) and *brittle* (i.e., easily broken when modified for any purpose). Perhaps worse is the widespread fear that legacy ISs will, one day, break beyond repair. Such fears combined with a lack of techniques or technology to fix legacy IS problems result in IS apoplexy. That is, legacy ISs consume 90% to 95% of all application systems resources. This prevents organizations from moving to newer software, such as client-server configurations, current generation DBMSs, and fourth generation languages (4GLs), let alone the architectures described above. Consequently, organizations are prevented from *rightsizing*, which involves moving from large mainframe computers to smaller, less expensive computers that fully meet current application systems requirements. This apoplexy, in turn, is a key contributor to the software crisis. New requirements, often called the IS backlog, cannot be met since legacy ISs cannot be extended and new systems cannot be developed with the 5% to 10% remaining resources.

Legacy IS migration involves migrating an existing IS into a target IS. This could mean replacing all of the hardware and the software (i.e., interfaces, applications, and databases). However, under some circumstances, some existing components can, even should, be incorporated into the target IS. I claim that the target environment should, in principle, be the distributed computing and enterprise information architectures, described above. In ICIS terms, legacy ISs can be incorporated into the enterprise information architectures as component ISs.

Even if these environments were in place, there are few if any methods for migrating legacy ISs to the new environment (one is proposed in [BROD92b]). Currently disjoint and heterogeneous information/computing resources must be made to cooperate efficiently and transparently. Without cooperation (e.g., via interoperability) and increased *intelligence* of these resources, the massive investment may be lost. For example, do you incorporate, into an organization's new distributed computing architecture, a mission critical, multimillion-dollar, multimillion-line COBOL system with all its faults and limitations, or simply replace it with a newly rewritten system? There is technical and economic evidence that legacy ISs cannot be re-written.

Legacy IS problems are much more compelling and immediate than the vision of distributed computing. No matter how great the vision, it will be of little value if it cannot be integrated into the current IS technology base. A challenge here is to develop technology that permits enhancement and evolution of the current, massive investment in ISs.

2.1. Legacy Information System Case Studies

This section illustrates legacy IS problems via actual legacy IS migration efforts, identifies solution directions, and restates distributed computing research challenges in terms of legacy IS challenges. The message of this section is that the visions described above inadequately address legacy ISs. Researchers and technologists should provide effective means to migrate from legacy ISs and the installed technology base to newly offered technologies. The good news is that the potential cost reductions of the next generation computing and the avoidance of the sins of the past will pay for the vast migration costs. In a competitive environment, you may not be able to afford not to migrate!

2.1.1. Telephone Service Provisioning

Figure 11 illustrates eight very large and very real legacy ISs used in the provisioning of telephone services to telephone customers. When a customer calls to request telephone service, this combination of ISs supports what I call the *service order transaction*. It consists of thirteen steps, each supported by one or more legacy ISs. The steps verify the customer street address, identify an appropriate (e.g., available on your premises, working, not too recently used) cable pair, telephone number, and related equipment (e.g., line equipment, location on a distributing wire frame, jumper wires, special circuits for advanced services, cross connect box) in their available inventory, assign the equipment to you, take it out of inventory, deploy it (i.e., make the necessary connections, update the telephone switch that will serve you), ensure that it works, and inaugurate the service. These steps invoke others not illustrated here, including customer credit validation, account and billing setup, and directory services update. Ideally, service provisioning is completed during the customer contact or within a few hours or days.

As with most legacy ISs, none were originally designed to cooperate with any other to achieve any task or goal. Since telephone service provisioning is a mission critical for the telephone company, the systems were made to interoperate using existing technology. There is no transaction system to support the design, development, testing, and execution of the service order transaction, so it is done using available technology without the value of high-level abstractions, locking mechanisms, transaction-oriented backup and recovery, performance tuning, etc. As you can imagine, such services might dramatically improve IS support for telephone service provisioning.

Major problems in constructing the service order transaction are due to the heterogeneity of the participating systems. They are almost all written in different languages and run on different computers under different operating systems. They all have different interfaces. In some cases, to interact with the system you must simulate a TTY terminal to act as an on-line user. Each system has a different information base with naturally arising semantic inconsistencies between definitions and representations of similar concepts (e.g., customer, service order, street address).

Most legacy ISs actually interoperate or have requirements to interoperate in a similar manner. The service order transaction is an excellent illustration of the need for improved technical support for interoperability such as raised in the previous section. It is a real example of *programming in the large* (e.g., the service order transaction involves operations on eight large and distinct ISs) for which there is almost no support such as advanced transaction models [ELMA92] that provide some, as yet undefined, notions of transactional integrity in multi-IS tasks, and no high level programming languages to support what are currently called workflows. With eight major ISs and thirteen steps, many things can go wrong. Transaction abort and restart may be infeasible due to the lack of control over component ISs or too costly. In the example, what is a reasonable definition for transactional integrity? What is required to support it?

Gateways provide potential solution to some problems of architecture, systems interaction and interfacing, and heterogeneity via encapsulation. Gateways can be constructed to mediate between the encapsulated IS and requesting IS [MANO92]. They can provide requesting ISs with appropriate views and provide a basis for interoperability. By encapsulating the IS, the gateway provides a means of hiding any modifications to the legacy IS (e.g., the evolutionary replacement of the legacy IS by one or more ISs). The gateway concept is used in all legacy ISs described in this section. Hence, research to facilitate the design, development, deployment, and modification of efficient gateways is critical to both legacy ISs and future distributed computing architectures.

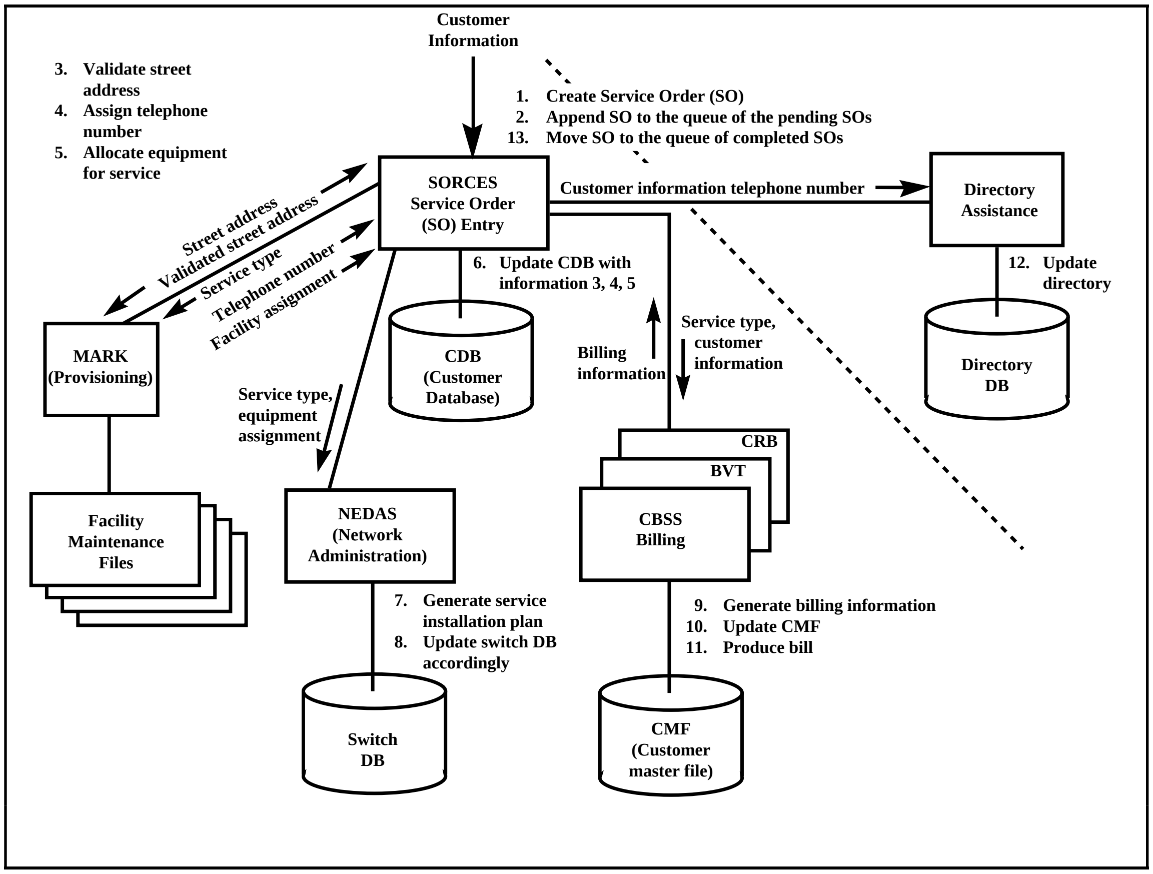
Customer Information
3. Validate street address
4. Assign telephone number
5. Allocate equipment for service
1. Create Service Order (SO)
2. Append SO to the queue of the pending SOs
13. Move SO to the queue of completed SOs
SORCES Service Order (SO) Entry
Customer information telephone number
Directory Assistance
Street address
Validated street address
Service type
Telephone number
Facility assignment
6. Update CDB with information 3, 4, 5
12. Update directory
MARK (Provisioning)
CDB (Customer Database)
Billing information
Service type, customer information
Directory DB
Service type, equipment assignment
CRB
BVT
Facility Maintenance Files
NEDAS (Network Administration)
CBSS Billing
7. Generate service installation plan
8. Update switch DB accordingly
9. Generate billing information
10. Update CMF
11. Produce bill
Switch DB
CMF (Customer master file)

Figure 11. Service order transaction.

2.1.2. Cash Management System

Like all other legacy ISs described in this section, the CITIBANK Cash Management System (CMS) was believed to be impossible to extend without massive risk and cost. Indeed, several attempts have failed for

CMS and for other legacy ISs described here. This sub-section summarizes a potentially successful attempt [BROD92b].

CMS supports check processing and other specialized services for large corporate customers. It allows a customer to maintain a so-called *zero balance* account (i.e., the bank notifies the customer of all the checks that are processed during a given day, and allows the customer to cover the exact amount of these checks with a single deposit). Hence, the customer applies the minimum possible capital to cover his liabilities and only when the capital is needed. Second, CMS *reconciles* cleared checks. A customer provides the bank with an electronic feed of all the checks written each day. The bank matches issued checks against cleared checks and provides the customer with an electronic feed of all cleared and pending checks. Third, CMS supports electronic funds transfers between customer accounts. When the initiator or recipient of the transfer is another bank, then funds must be electronically received from or transmitted to another bank. This requires connection to several electronic money transfer systems (e.g. Swift, Federal Reserve Bank). Fourth, CMS supports *lock box* operations for customers who receive large numbers of checks in the mail, such as a large landlord or a utility. Mail received in a post office box is opened, the checks deposited for the customer, and an accounting rendered. Such a service is appropriate. Finally, CMS supports customer on-line inquiry and reporting of account status as well as on-line transactions such as the previously discussed transfer of funds.

CMS encompasses 40 separate software modules that perform these and other functions, totaling around 10^7 lines of code. Most of the code runs in a COBOL/CICS/VSAM environment; however, the connection to the Federal Reserve bank is implemented on a Tandem machine using TAL, and lock box operations are provided on a DEC VAX.

The majority of the system was written in 1981, and it has now grown to process 10^6 checks in a batch processing run each night and approximately 300,000 on-line transactions each day. The majority of the systems run on an IBM 3090/400J with 83 spindles of DASD. CMS must continue to operate 24 hours a day, 7 days a week.

The key objective of the migration was that it be an iterative evolution, which we call *Chicken Little*, rather than a complete one-shot replacement, which we call *Cold Turkey*. Management imposed the criteria that each iterative step require less than ten person years of effort, one year in duration, and one further year to payback. The following migration plan was proposed.

1) Peel the Onion: Peel off successive layers of a complex system until only the *core* remained. Upper layers could be moved to a new environment in manageable sized *chunks*. On-line reports and feeds are especially amenable to this treatment.

   In CMS, we were left with a small *core* which had manageable complexity. It is our assertion that most complex systems can be peeled in this fashion. If CMS had been poorly architected, for example, if CITIcash had performed its own updates instead of calling CITIchecking, then the *core* would have been larger. In this case, we speculate that re-engineering of multiple kernels into a single kernel would have been the appropriate step. This was not required in CMS.

2) Decompose Functionality: When a complex IS implements two or more functions that are logically separable, then the migration plan should untangle the multiple functions and migrate them independently. This is especially valuable when migration steps would otherwise be too large to perform.

3) Design the Target: The target system was designed as a Cash Management ICIS composed of less than ten component ICISs to run on a distributed computing architecture. The resulting design is the target of the migration strategy.

4) Migrate: Completely rewrite the selected components, one at a time, migrating the data and functionality, thereby incrementally constructing the target system.

   The key element of the migration plan is that it is an incremental rewrite of CMS and not incremental re-engineering. Although there has been much interest expressed in re-engineering legacy ISs, our case study has indicated that virtually all code would be better re-specified using modern tools, especially fourth generation languages, report writers, and relational DBMS query languages. There may be legacy ISs where re-engineering is a larger part of the migration plan; however, our experience did not indicate any significant use of this technique.

Several research areas emerged from this legacy IS migration exercise [BROD92]. First, and foremost, a gateway was seen as critical to legacy IS migration, but its performance was seen as more critical than the other gateways used in other migrations described in this section. Another research area is support for database migration (e.g., from legacy to new) and application cutover. A third area was tools to analyze and extract systems specifications, database designs, and the logical structure of application code. A final research area is an environment for distributed IS design, development, and migration to support not only legacy IS migration but continuous evolution of ISs, ICISs, and support technologies.

2.1.3. Facilities Management System

Many legacy ISs have evolved from simple systems, by adding a small amount of functionality at a time, to become massive systems (e.g., millions of lines of COBOL). Fifteen to twenty years is ample time to embed every conceivable IS blunder deep into an IS. Such ISs are seldom documented. FMS, a telephone facilities management system, is such a legacy IS. I focus here on only a few of the problems and a proposed solution. The vast FMS database is a much more valuable resource than FMS functionality. Many existing systems (i.e., 40 critical systems, over 1,200 lesser systems) depend on accesses to FMS. Many new ISs require access to the data. However, due to the inflexible data structures and the systems interface, the data is largely inaccessible. As a result of this and the construction of new ISs over time, between 40% and 60% of the data is duplicated in other systems. FMS must be in continuous operation, 24 hours a day, 7 days a week.

How do you migrate FMS from its current legacy IS to a facilities management ICIS in a client/server distributed computing environment, following the distributed computing vision? The proposed migration strategy is as follows: First, construct a gateway to encapsulate FMS and the new IS so that changes are transparent to ISs that depend on FMS. Figure 12 illustrates the migration architecture, including the legacy user (UI_i) and systems interfaces to the legacy IS and the new user interfaces (GUI_i) and applications modules (M_i) on the new DBMS. Second, design new databases that include necessary FMS data as well as meet current requirements. Due to FMS' structure and a lack of documentation, this will require, in part, treating FMS as a black box and studying its behaviour externally. Populating the new databases is a challenge. Slowly migrate FMS functionality from FMS to the new databases using the gateway to direct requests to the appropriate ISs (e.g., old FMS, the new IS, or both). Due to potential internal dependencies in FMS, it may be necessary to continue to maintain FMS in its full form even when functionality has been migrated to new systems. Eventually, throw FMS away.

Under cover of the gateway, any IS architecture can be used, including ones consistent with the distributed computing vision. The requirement to support over 1,200 legacy ISs emphasizes the importance of gateway research and technology. A major function of the gateway is to direct requests to the correct system. This provides an opportunity for an intelligent gateway to add view and query optimization. However, powerful transaction management support must be added. There is no choice. Interoperability research should consider such legacy IS migration requirements. Finally, in this legacy IS migration, as with all others in this section, object-oriented technology was not explicitly used due to the lack of adequate concepts, tools, and techniques, let alone experienced staff. However, the migration architecture, the use of message passing between components, and component encapsulation are appropriate steps toward future conversion to object-orientation and a distributed computing architecture. This illustrates the relationship between the vision of future computing, discussed in Section 1, and legacy systems. Legacy systems are intended to be encapsulated component ISs in an ICIS.

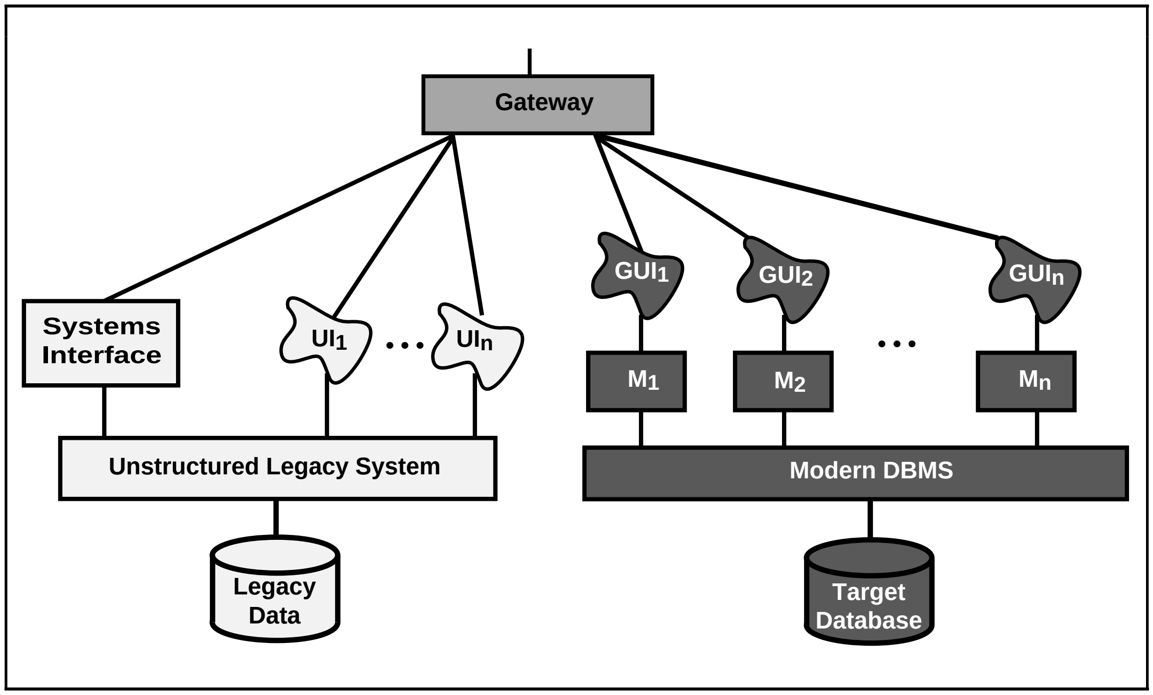

Figure 12. Legacy IS migration architecture.

2.1.4. The Corporate Customer Database

US West, a large American telephone company, has over 1000 ISs that deal with customer operations, almost none of which were designed to interoperate [HDBSI92]. Each has its own customer *database*, however over 200 are major customer databases. This is typical of large telephone companies around the world. There are myriad legacy IS problems in such a nexus of customer operations support systems. Most of the systems are inflexible (e.g., data structures cannot be enhanced or modified) and fragile. Bitter experience has taught that large legacy ISs cannot be rewritten from scratch, the cold turkey approach. Stories of the failures of multimillion dollar, multiyear rewrites abound. Organizations feel that "You can't live with them and you can't live without them."

In recent years, many large organizations have investigated the idea of corporate information repositories, also called corporate subject databases. This suggests that all customer data be integrated logically, and possibly physically, into one corporate customer database. The conceptual modelling world has offered the global conceptual schema approach. The distributed database community has offered limited distributed database interoperability. To date, most such projects have failed. Each database has its own definition of *customer*. The definition is used and depended upon by many applications. Even if the definition could be altered so that the data could be migrated to the new database, there would be a massive problems of dealing with the old applications. Rather than one definition of customer, there are good reasons for contextual variations (e.g., regulatory, legal) that are inherently inconsistent. The scale of the customer database completely defeats all proposed global schema, integrated schema, and conceptual modelling solutions. The scale of the ICIS and world-wide computing puts the nail in the coffin. Even if we had the tools with which to conceptually map schemas, the scale of the problem is beyond the manpower possible to be deployed to the task.

Consider semantic challenges that arise in making 200 customer databases interoperable. Interoperability requires that establishing, representing, maintaining, and possibly evolving through time the relationships between interoperable objects. Consider only the concept of *customer*. At the type or schema level, the above example involves at least 200 different definitions of *customer* in the databases and 1000's of different definitions in the applications. What should be the relationships between these definitions? At the individual level, what are the actual relationships between the 10^7 customers? Although many definitions may be different, the 1000 systems are highly redundant. Further, due to the nature of the business, many geographically distributed business customers are treated as separate customers (i.e., have separate accounts) in the same IS and in separate ISs. How do you establish the relationships between individuals? Type information can help but only to a limited degree. Further, semantic information such as we would like at the type level is often missing completely from legacy ISs or is distributed throughout the data and applications.

These legacy IS examples help to focus the earlier identified interoperability research efforts. Such experiences led to the hypothesis of no global anything and the need for more effective means for systems mapping tools such as type systems instead of conceptual models. Rather than pursue the infeasible and costly goal of complete integration, research should identify different forms of interoperability (e.g., powerful transaction and queries that achieve the required interactions). The example also emphasizes the importance of global naming schemes since it is a bad business practice not to be able to find a customer's records, regardless of the IS in which it is stored. Names are the minimum information that needs to be globally available. Global naming poses major problems since most legacy ISs do not have logical or flexible naming schemes, if any, and systems that do are all inconsistent. Global naming schemes must address legacy ISs or they will not be global.

2.1.4. The Repository

A particularly popular current trend in large IS shops is the construction of a repository, in the sense of IBM's AD/Cycle Repository and DEC's Cohesion. Amongst other things, a repository is intended to support an enterprise model, a design model, and a technology model for all ISs in an enterprise. This is sometimes interpreted to include a corporate-wide dictionary or directory. This brief example is intended to question the feasibility of such a repository and illustrate the principle of no global anything, raised above.

A common approach to building a repository is to integrate every schema element in existing ISs into an enterprise model or dictionary. For example, all definitions of *CUSTOMER* would be resolved (i.e., related) and possibly integrated into, for example, a generalization hierarchy. This incurs the problems raised above concerning semantic aspects of interoperability. In this context, resolving elements involves establishing, representing, maintaining, and possibly evolving through time, the relationships between those objects. In one corporate repository development, 27,000 schema elements from 40 major systems, of more than 1,000, are being resolved in this manner. The first step of establishing relationships is currently taking one person day for two elements. At this rate (and assuming e.g., no resolution tools or techniques), the first-step of this partial corporate repository will take 65 person years!

2.2. Migration Challenges

Migrating from the current installed technology base involves the migration of two aspects. First, the existing systems technology and its architecture must be migrated into the systems technology and architecture for distributed computing, as described in Section 1.2.1. Second, legacy ISs must be migrated, as described in Section 2.1. Both aspects of migration were studied in GTE. The resulting observations, summarized below, were compared with experiences in other large corporations and were found to be universal for large legacy ISs. The universal recommendation for addressing these problems was incremental evolution.

2.2.1. Legacy Information Systems Migration Challenges

The following is a list of the most common and important legacy IS migration challenges. The more mission critical an IS is, the more severe the problems. Cold turkey rewrites do not work. There is no clean sheet of paper. You must deal with the existing ISs, management, operations, technology, budgets, environment, people, etc. Half of the problems below (3, 5, and 7) concern embedding the new IS into the existing environment. These problems vastly complicate cold turkey replacement. The other problems (1, 2, 5) involve ensuring that the new IS captures all the functionality of the old IS.

1) That's All Of It, I Think — The development of large, complex ISs requires years to accomplish. While the legacy IS rewrite proceeds, the original legacy IS evolves in response to maintenance and urgent business requirements. It is a significant problem to evolve the developing replacement IS in step with the evolving legacy IS.

   More significant than maintenance and minor *ad hoc* changes are changes in the business processes that the system is intended to support. These are typically in a constant state of flux. The prospect of incorporating support for the new business processes in the replacement system may lead to significant changes to the system's purpose throughout its development. This dramatically increases the risk of failure.

2) Incomplete Specification Leads To Incomplete Functionality — The requirements for the old IS are never complete and are almost impossible to define. Many requirements have been met directly by coding solutions into the old IS without documentation. The old IS is the only real specification. (See also problem 5.) There is never a specification of the legacy IS due, in part, to the constant evolution of requirements and IS changes. Requirements that couldn't be met are replaced with approximations and work-arounds that are seldom documented. The standard life cycle does not have as a deliverable "A complete specification of the current IS."

3) Ripple Effect Problem — An IS that is mission critical naturally invites other ISs to connect to it. When you change the mission-critical IS, you must deal with all ISs that connect to it. How do you embed the new IS into the operational environment? For example, you can't replace one old IS with one or more standard ISs that cover the same functionality if they don't meet the requirements of the old IS provide on which other ISs depend. The more central the IS is to the organization, the greater the ripple effect.

4) Rebuild the Organization — In supporting some business requirements of some organization, an IS mirrors its structure, to some degree. Over twenty years, both the organization and the IS evolve. The relationship between them is complex in terms of business functions and politics. This poses nontechnical barriers to changes to individual ISs, let alone many ISs that might form an ICIS. These problems tend to overwhelm technical challenges.

5) The Outer Tar Baby: Dependencies To External Systems — Throughout the life of the old IS, vast numbers of small procedures or local systems have been developed to depend on the old IS (e.g., utilities, analysis and report programs). Many, if not all of these have never been documented or identified. First you must find them and then handle the requirement. In one of the above case studies, we found 1,200 undocumented utilities/small systems that use one mission-critical IS.

6) We Want It All — Users will not be satisfied with a new IS unless it offers them substantially more than the old IS. Cost justification and other organizational incentives argue for more than the old IS provided. It is easier to justify continual, expensive fixes than a costly rewrite with projected annual savings.

7) Jus' Load 'Er Up: Migration and Data Conversion — Once the new IS has been successfully implemented, you must migrate it into the operational environment. This requires that all of the data in the old IS be converted into the format of the new IS and that the new and/or old IS continue to support its functions. Once the new IS is fully loaded, it must be embedded into the existing operational environment. All this must be done without interrupting the mission-critical IS which must be available 99% of the time. How long does it take to download a terabyte database? (Also see problem 3.)

8) The Inner Tar Baby: Dependence Without Modularity — Like so many pre-database systems, the code and data of large, mission-critical ISs are intimately bound together. There is virtually no data independence. The system code and data is not modular, hence it is difficult or impossible to identify a subset of data or functions to extract or address independently from the rest of the system. Even though current documentation can, and often does for expository reasons, describe the old IS in logical groupings, the code is not so structured.

2.2.2 Legacy Information Systems Technology Migration Challenges

Legacy ISs were designed to be supported by the existing, hence, legacy, systems technology and architecture (e.g., simple flat file systems, ridged hierarchical DBMSs, COBOL, TTY interfaces). The new systems technology and architectures (e.g., described in Section 1) may not support legacy ISs, and *vice versa*. I see the following problems as some of the most significant problems with the current systems technology base. Although the new technology base may attempt to avoid such problems, there are no known means for migrating from the old to the new, except for cold turkey. Evolutionary, incremental means must be developed.

- Data Liberation — Users and systems cannot easily access or store the necessary, often already existing, information as it is bound to applications in multiple systems.

- Inability of Systems to Interoperate — Systems do not easily or adequately interoperate (e.g., batch vs. on-line/real-time; tasks involve multiple systems that must interact).

- Systems Designed to Be Inflexible — Poor design and development, older technologies, and techniques have resulted in ISs that are difficult to maintain, evolve, and enhance, due to inflexible systems design and development; inter-system dependencies; lack of modularity; lack of access to data; and business rules and policies (the way business is done) embedded in ISs in inaccessible and fixed representations (e.g., difficult to find or change the rules governing billing).

- Inadequate Life Cycle — Current life cycles include requirements gathering, specification, design, development, testing, implementation, maintenance, enhancement, and evolution. They do not adequately address current problems and the long-term requirements. There is almost no life cycle support (e.g., environments or methodologies) for continuous evolution of ISs that last twenty years or more. Past history says that you build a large, mission-critical IS only once. Thereafter you must support its evolution. We should build ISs to last 100 years!

- Diversity of Information Bases (including data and function definition) — This is a data administration and standardization issue. For reasons such as funding methods, a lack of knowledge of exiting systems and data, and inflexible systems (e.g., difficult technical issues in migration, conversion and cutover), there has been a proliferation of independent systems and information bases. In the past, standards were avoided. This has led, in part, to the diversity.

- Diversity of User Interface and Presentation Formats, Tools, and Technology.
- Inadequate System Responsiveness — Current systems performance does/will not meet real time needs. Today's standards concern screen presentation times. They should focus on ensuring that the intended function, or any related to it, meet business requirements to be accomplished within reasonable limits. This means that the right information is accessed and presented in a way appropriate to the viewer and the task.
- Costs Invested In Existing Systems Technology Base — The massive investment in the existing systems technology make modification difficult to justify.
- Proliferation of Independent Systems and Data.
- Inadequate Skills Match To Meet Current and Future Systems Technology Requirements.
- Lack Of Focus On Infrastructure In Systems Technology — Legacy IS projects acquired and used their own systems technology. The new view is of distributed resources to be shared by all ISs in the distributed computing environment and supported by a corporate-wide system technology infrastructure. This major change in systems technology leads to radical changes to funding and administration of IS. Consequently, it poses major, nontechnical challenges.
- Inadequate Management Information, Controls, Measures, and Processes — Current metrics (e.g., reliability and up-time) are not adequate to meet current or future business requirements. There is a lack of management information to adequately manage distributed computing environments. There are inadequate tools to manage the versions required in the continuous evolution of large-scale ICISs, including all legacy IS migrations.

As you can see from the above problem list, the legacy problem is far more than the migration of legacy ISs. Legacy ISs depend on legacy IS technology, which depends on legacy concepts, tools, and techniques, which depend on legacy management, which all depends on legacy thinking which often tries to achieve homeostasis. Much of computer science is based on assumptions that are no longer true (e.g., Von Neuman machines, network communications as bottlenecks). The value of the legacy can be argued, but the problems, when you want to move on, can seem to be insurmountable.

3. Killer Applications For World-Wide Computing

The challenges we face in realizing world-wide computing and ICISs are far more than technical. The technical challenges are exciting, but they do not pose the greatest challenges. Legacy problems, ranging from technology to thinking, are also not the greatest challenges. The greatest challenge is to find ways in which computing can solve major human problems and meet compelling human needs.

In telecommunications, computing, and the potential integration of the two, we have solutions in search of problems, on one hand, and a lack of solutions for existing hard problems on the other. The visions are not new. They are progressing very slowly. Again, the central problem is not the technology nor our legacy. The technologies do not yet meet a human need or solve a problem in a way that people would pay anything for. Originally, databases were hard to sell since they violated ways of doing business (e.g., shared data means loss of ownership and control). Distributed databases pose similar threats today. Databases are more acceptable as support for applications within divisions, when ownership is not lost. For nontechnical reasons, amongst others, the original vision of corporate ownership of data is far from reality.

It takes a killer application to overcome the legacy. We have no creative, compelling applications for world-wide computing or ICISs. Lotus 1-2-3, almost on its own, started the PC/minicomputer revolution, which led to the movement of applications to the desktop and the economics (i.e., cheap workstation MIPS) that is leading to distributed computing and the demise of mainframes. Lotus 1-2-3, for less than $500, motivated managers to buy $5000 machines. If I were to offer you everything you currently have on your desktop plus a lot more, for a lot less cost, but on a mainframe, would you take it? I haven't met anyone who has said yes. Hence, the key point was not cost, nor the technology, but rather what Lotus 1-2-3, the application, brought, namely personal autonomy, control, and power which led to personal innovation. It facilitated real work and met real needs in ways that people would pay for. That's what makes revolutions. For both next generation computing and telecommunications, we have no such killer applications.

What do killer applications look like and how do you discover one? They should meet some significant human, business, or societal need in ways that are compelling or even just acceptable to those with the need. Hence, to find it you must understand human or organizational needs. Computer scientists may not be as well suited to this as application domain experts. Computing and communications must be critical enabling elements. Computer scientists can help here. To find killer applications, hence to assist in realizing the visions, you should understand applications and interact with application experts, those people and organizations with the problems.

My current guess is that killer applications may not be individual applications but the result of multiple, possibly pre-existing applications, working in cooperation (e.g., not just spreadsheets, databases, text processors, schedulers, billing programs, etc., but some very useful combination of these). Correspondingly, future technical successes will come not from individual technologies but from the cooperation of multiple technologies.

To get over the legacy, to find killer applications, we may have to use new ways of thinking, new perspectives. I will conclude with one currently popular such method called Process Re-Engineering.

Conventional ISs are designed to support specific functions of some organization. For example, in the service order transaction example, described in Section 2.1.1, eight ISs each provide a specific function. The problem with the service order transaction was that the systems were never built to cooperate. There was little focus on the higher level function that they now collectively support. The perspective was a bottom-up view of the business, focusing on specific functions.

In process re-engineering, the focus is exclusively on the critical business processes, the lifeblood of your organization. In the above example, it is telephone service provisioning. Hence, the service order transaction is important rather than the supporting systems. With this orientation, concern is for the process to ensure that it goes smoothly from beginning to end, and for the role the business process plays in the organization. In turn, we can view such processes as placing requirements on the supporting ISs. When business processes change, the IS requirements change. Function-specific systems may no longer be of use. Hence, IS technology must be considerably more flexible than it is today. Process re-engineering encourages a new way of thinking of ISs. Perhaps a killer application is one that permits the constant recombination of arbitrary IS components to meet the evolving requirements of ever-changing business processes.

It might be possible to imagine extrapolations of current ISs to intelligent and cooperative ISs in a variety of domains, such as health care. It might also be possible to imagine extrapolations of current telecommunications (e.g., plain old telephones, FAX, modems) to the communication of any information, in any form, at any time, to any location. But is beyond my power to imagine the potential of world-wide computing/communications and what contributions it might bring to make the world a better place.

The ideas presented in this paper can be summarized in one sentence. *The greatest challenge for future ISs and IS technology is their capability to accommodate change.*

ACKNOWLEDGMENTS

The author gratefully acknowledges the insightful discussions with Sandra Heiler and Frank Manola concerning topics in this paper.

References

[ARM89] *The ANSA Reference Manual*, Architecture Projects Management Limited, Poseidon House, Castle Park, Cambridge, U.K., 1989.

[BEER90] Beeri, C., "A Formal Approach to Object Oriented Databases," *Data & Knowledge Engineering*, 5 (1990) 353-382.

[BROD92a] Brodie, M.L. and S. Ceri, "On Intelligent and Cooperative Information Systems," *International Journal of Intelligent and Cooperative Information Systems* 1, 2 September 1992.

[BROD92b] Brodie, M.L. and M. Stonebraker, "DARWIN: On the Incremental Migration of Legacy Information Systems," DOM Technical Report, TM-0588-10-92-165, GTE Laboratories Incorporated, November 1992.

[BROD84] Brodie, M.L., J. Mylopoulos, and J.W. Schmidt (eds.), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag, New York, February 1984.

[CARR89] Carriero, N. and D. Gelernter, "Linda in Context," *Comm. ACM*, 32, 4, April 1989.

[ELMA92] Elmagarmid, A.K. (ed.), *Database Transaction Models For Advanced Applications*, Morgan Kaufmann, San Mateo, CA, March 1992.

[FONG91] Fong, F., et. al. (eds), "X3/SPARC/DBSSG/OODBTG Final Report," interim draft of 17 September 1991.

[GRAY93] Gray, J. and A. Reuter, *Transaction Processing: Concepts and Techniques,* Morgan Kaufmann, San Mateo, CA, © 1993, published October 1992.

[HEIL90] Heiler, S. and S. Zdonik, "Object Views: Extending the Vision," *Proc. 6th Intl. Conf. on Data Engineering*, Los Angeles, Feb. 1990.

[KAPL92] Kaplan, B.A., et al., "Communicopia: A Digital Communication Bounty," Investment Research Report, Goldman Sachs, New York, New York, 1992

[MANO90] Manola, F. and A. P. Buchmann, "A Functional/Relational Object-Oriented Model for Distributed Object Management: Preliminary Description," TM-0331-11-90-165, GTE Laboratories Incorporated, December 31, 1990.

[MANO92a] Manola, F., M.L. Brodie, S. Heiler, M. Hornick, and D. Georgakopoulos, "Distributed Object Management," *Int'l Journal of Intelligent and Cooperative Information Systems* 1,1, April 1992

[MANO92b] Manola, F. and S. Heiler, "An Approach To Interoperable Object Models," Proceedings of the International Workshop on Distributed Object Management, Edmonton, Canada, August 1992.

[OMG91a] Object Management Group, "The Common Object Request Broker: Architecture and Specification," OMG Document Number 91.12.1, Draft 10 December 1991.

[OMG91b] Object Management Group Object Model Task Force, "The OMG Object Model," draft 0.9, OMG Document Number 91.9.1, September 3, 1991.

[SCHI89] Shilling, J.J. and P. F. Sweeney, "Three Steps to Views: Extending the Object-Oriented Paradigm," in N. Meyrowitz, ed., *OOPSLA '89 Conference Proceedings*, ACM, Oct. 1989, *SIGPLAN Notices*, 24(10), Oct., 1989.

[HDBSI92] *Proc. 1992 Workshop On Heterogeneous Databases and Seamn tic Interoperability*, Boulder, CO, Feb. 1992.

[WILL91] Williamson, G.I. and M. Azmoodeh, "The Application of Information Modelling in the Telecommunications Management Network (TMN)," *Proc. Telecommunications Information Networking Architecture Workshop* (TINA91), March 1991.

[ZATT92] Zatti, S., J. Ashfield, J. Baker, and E. Miller, "Naming and Registration for IBM Distributed Systems," *IBM Systems Journal*, 31, 2, 1992.

Appendix: A PERSONAL STATEMENT
Computer Science and Concern for Our Planet

This message comes from my heart, from my spirit. Please consider with me some challenging questions for which I have no clear answers but which provide a source of creativity and inspiration in my professional and personal life:

- What kind of world are we creating with computers?
- What kind of world do you want to live in?
- How can *you* use computers, amongst other things, to create that world?
- Are you doing those things now? Or, at least, is your work consistent with your desired world view?
- Whatever you are doing with computers, you are changing our world. Is it for the better?

Computers provide an enormous power on our planet. Moment by moment their power affects and influences you, me, nations, the world economy. Their power and influence will continue to grow enormously. Like all potent powers, computers can be used in many ways. As computer scientists, we play a key role in directing the future use of that power.

Computer scientists often accept jobs or funding to achieve tasks without considering whether these jobs or tasks are in keeping with their desired world view. At least, it is difficult for me to imagine that they want their children to live in the world that could result from their work. Such inattention to values and goals leads to monsters, waste of minds, waste of money, or just plain poor computer systems.

There are also computer scientists and detractors of computer science whose world views exclude many or even all applications of computers. Such myopia can prevent the *positive* deployment of the amazing power of computers. Computer scientists with this positive attitude may be unable to obtain funding for creative, positive applications of computers.

Where do *you* fall in this spectrum of concern for the effects of your work? What kind of world are *you* creating? Do you feel good about *your* work?

I have grown a great deal professionally and personally, in part, by considering these issues. My goal is to align my intellectual, spiritual, and physical beings. I want everything I do, without exception, to contribute to a world that I want to live in. Such a vision has provided me clear direction concerning what I want to do with computers and what I refuse to do with computers. A major consequence of the process is dramatic freedom of thought, inspiration, and creativity. I am more creative now than I have ever been before in my life. I'm having a blast!

I encourage you to align your mind, spirit, and body to consider vigilantly how your work contributes to improving the world for all beings, to find the strength to refuse tasks that do not contribute to your world view, and to delight in the creativity that will come from this alignment. Each of us is capable of these things and much more! Our work in computer science is as important in its way as that of Mother Teresa. Let us strive to make our work worthy of such a comparison.