THE CASE FOR OPEN SYSTEMS

Gary J. Nutt

CU-CS-515-91 February, 1991

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309-0430

nutt@cs.colorado.edu

ABSTRACT

Open systems is a buzzword phrase of the nineties, used to describe various aspects of computer systems made up of components from various vendors. In some cases, the phrase refers to the interoperability among the heterogeneous components, while in other cases it refers to software portability or integrated systems.

The critical aspect of contemporary computer systems is the requirement that the system be made up of many different components -- distributed systems -- and that the components be able to be used together to support the corporate computational needs.

In this paper, abstracted from a forthcoming book,[1] we examine the evolving corporate computer environments to understand why open systems are currently critical to the corporation. The paper also explains why *interoperability*, *portability*, and *integration* are all critical aspects of open systems.

1. G. J. Nutt, *Open Systems*, Prentice Hall, to appear 1992.

1. INTRODUCTION

Today's business world depends upon complex computer systems composed from personal computers, workstations, departmental computers, mainframes, and interconnecting networks. Computers no longer occupy just the back room of the organization, but are accessible on professionals' desks to serve as an integral tool to query, review, and update the global information in the organization's database. Besides acting as the agent for observing and manipulating the organization's information, the computer is used for direct correspondence, publication, decision support, planning, and tracking. The computer is becoming the fundamental tool to assist the information worker in fulfilling his or her duties.

As the computer and its associated network becomes more widely-used in various organizations, it must be designed to be succeedingly more accommodating of the needs and desires of the individuals who use it. Accommodating systems must provide information in familiar media representations (voice and images in addition to text and numbers), must provide a model of interaction with the users that is familiar or easy to learn, and provide a broad spectrum of functionality.

These demands on computer systems tax the resources of even the largest, most-successful computer manufacturers. The system must provide leading edge technology in all sizes of computer, it must provide full interconnectivity among the parts using high speed communications networks, and it must be facile with multimedia representations. And the system should be available to the end user organization at the most competitive price.

As a result, today's computer systems are designed and constructed as a composition of parts (see Figure 1): Individual computers have controllers, devices, memory, and software from vendors other than the one that manufactured the skeletal system. Software, in turn, is continuing to be constructed from collections of reuseable modules. At the network level the system introduces additional complexities: Systems are built from collections of distinct machines, each possibly manufactured by a different vendor, connected by a telecommunication network.

How can one ensure that such a collection of parts can be successfully connected together? How can one know if the parts will work together even if they can be connected? How can one assure that the

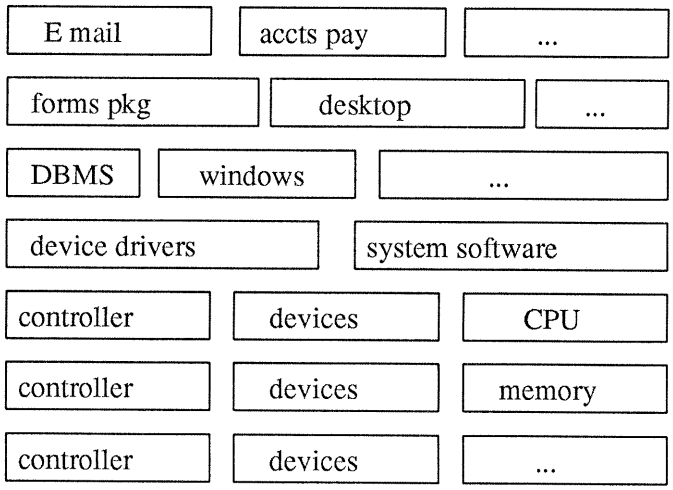

Figure 1: Systems Built from Components

synthesized system performs the intended function? How can a component supplier produce parts that are useful in many such configurations? How will end users know how to make good use of the technology that is provided?

This is the challenge of building contemporary computer systems and system components (such as workstations, servers, and software). Contemporary computer system design is an exercise in synthesis: The end system is a composition of hardware components, software components, and network components. Because of the breadth of technologies spanned by these component domains, it is not generally practical to construct an economical, high-performance computer system from components provided exclusively by a single manufacturer. (In those few cases where it is possible for a single company to provide all of the components, then it is necessary for different divisions of the company to provide various components. Effectively, the system is constructed from component produced by diverse organizations, and thus has a similar synthesis problem to that confronted by smaller system integrators.)

Nor is it in the best interest of the consumer to attempt to purchase complete corporate information systems from a single vendor. The consumer requires a custom configuration, with particular technologies accounting for more or less importance, depending upon the organization's requirements. For example, support of voice data may not be of high importance to an organization in its current short-term plan, while the business depends upon wide availability of images for all of its information workers.

From the consumer's point-of-view, such a situation ultimately leads to a market in which there may be competition among component providers, encouraging better-performing components at low costs. However, the components must interconnect and operate with one another properly.

The *open systems* approach to systems design and configuration is a reaction to this complex world. An open system is specifically designed to accommodate components from various vendors.

As suggested by Figure 2, we can view open systems from three distinct perspectives, depending upon the particular reason for studying the system: portability, interoperability, and integration. Each of these views focuses on a different aspect of composite systems interconnection, and each is a fundamental cornerstone of a successful open system.

Portability is the aspect of a system component that allows it to be used in various environments. As illustrated in Figure 3, in open systems, software portability is a major issue, since the software provides the basic functionality desired by an end user organization. To the extent that a software application program can be ported from one hardware system to another, then the application can be more profitable to its creator and more useful to end users that own equipment from a variety of hardware vendors. This is the view used by X/Open, OSF, UI, and other organizations interested in encouraging situations in which software can be adapted to many different vendors' hardware offerings.

Interoperability (Figure 4) refers to the ability of individual components within an open system to exchange information. At the lowest level, this implies communication subnetwork technology, and at the higher layers, it refers to peer-to-peer operation among programs on different computers. From the perspective of open systems, interoperability refers to effective information exchange at the peer-to-peer level. The interoperability aspect of open systems is most often emphasized by organizations that focus on network and interconneciton technologies, e.g., the Open Systems Inteconnection (OSI) unit of the International Standards Organization (ISO).

The *integration* aspect of open systems is concerned with the consistency of the various human-machine interfaces between an individual and all hardware and software in the system, see Figure 5. Thus, integration addresses human factors and cognitive models of operation. This view of open systems is at the heart of user interfacc discussions, particularly relating to graphic user interfaces (GUIs). Underlying the concern for user interface design is end user acceptance of products; to the extent that they can

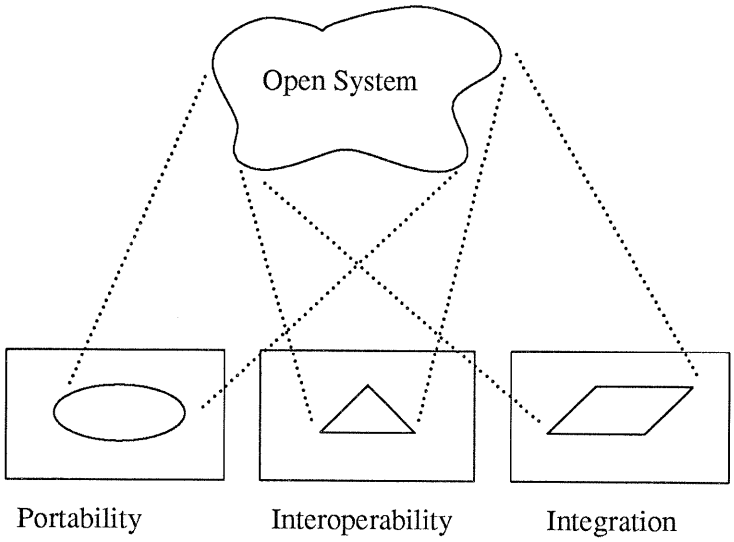

Figure 2: Open System Perspectives

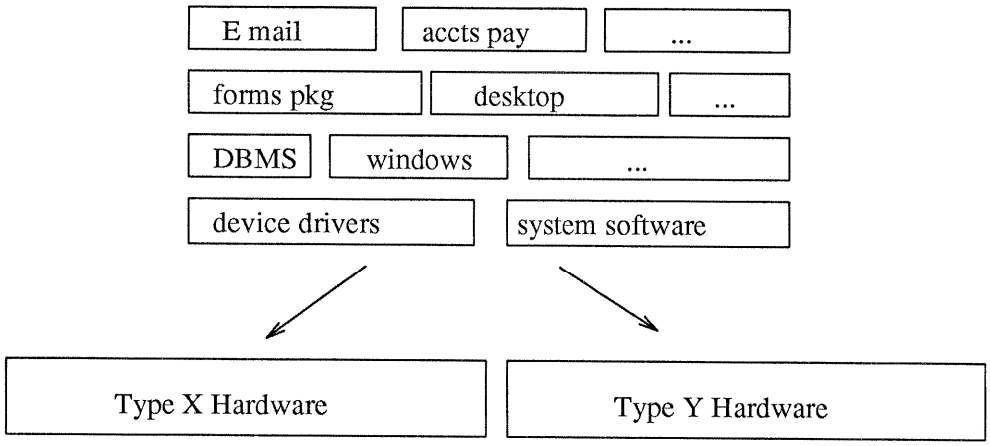

Figure 3: Portable Software

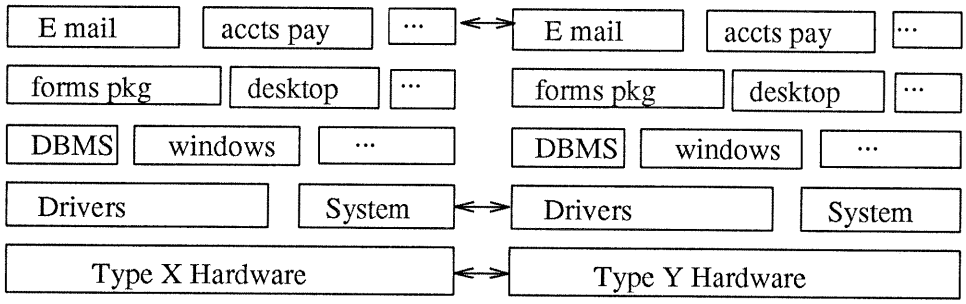

Figure 4: Interoperable Systems

be made consistent and intuitive, it will be easier to market such products into the end user world.

Open Systems Definition

There are a number of definitions used for open systems by different organizations, depending upon their point of view. Hardware vendors and software vendors tend to focus on supply-side aspects, while standards and user organizations emphasize the demand side perspective. Our definition of open systems attempts to reconcile the views.

Composite systems are molded together under some "grand plan" in order to meet the requirements imposed on the overall system. The architecture defines the grand plan:

> A system's *architecture* describes the structure of interconnection of the parts in the system. The architecture specifies component interactions, and ultimately the details of the *interface* among interoperating components.

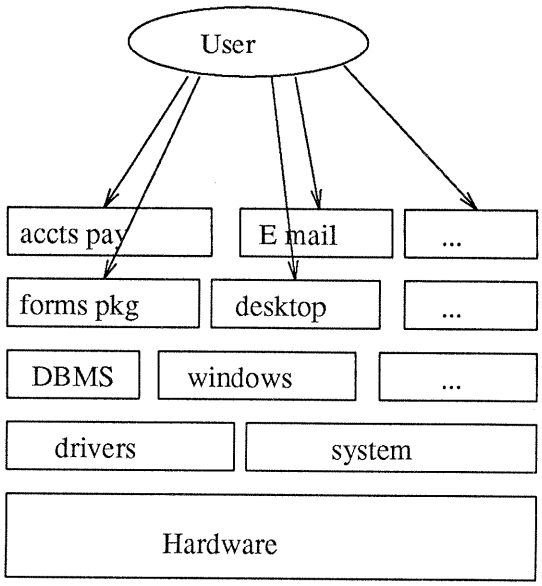

Figure 5: Integrated Systems

Traditionally, the architecture has been a proprietary part of a vendor's system, since the vendor perceives value in the means by which it partitions functionality then manufactures the systems based on that partition. For example, several vendors might produce a floating point unit, where an important difference among the units is the parallelism among floating point operations: In some cases, the unit is partitioned into an array of adders and multipliers, while another vendor incorporates a single pipelined unit. Each is likely to view the crucial aspect of his product as being the architecture, thus the commercial success of the product would depend on the architecture rather than the manufacture, sales distribution, or other part of the business.

For a computer system, the arguments for proprietary architecture are less clear than they are for a floating point unit. A computer system includes devices, controllers, memory, an operating system, a database system, a window system, and application programs. While the floating point unit is packaged as a single item, computer hardware and software systems are configurable by the user. Thus, computer systems offer many more opportunities for configuration of a particular instance of a product than do floating point units.

If the architecture is proprietary, then the user will necessarily purchase all components of the system from the vendor. Today, system users typically require that the system be open for competition, at least in terms of the manner in which components are composed in the system, i.e., the users require that the architecture not be proprietary:

> If the architecture is agreed upon by all of the component producers for the system, then the parts can be interconnected; such architectures are *open system architectures.*

Computer systems architectures are inherently *layered* systems in which components are produced from lower level components. For example, logic circuits are used to construct integrated circuits, integrated circuits are used to construct subassemblies, and subassemblies are combined to form boards, and boards to form computers.

Similarly, software is usually layered, at least as "systems software" and "application software." We conceptually visualize computer systems as being a sophisticated, layered architecture similar to that shown in Figure 6.

Layered models of the architecture hide details of implementation among unrelated components. For example, the implementation of "Programming Tools" in Figure 6 should not depend upon the implementation of "Boards" in the hardware, even though both are important parts of the computer system. Instead, the functionality of the system is partitioned so that there are low level function and high level functions, where the higher level functions are designed in terms of the interface to the low level functions, *but not their implementations.* Hardware designers have worked in this environment for years, and Dijkstra and other software designers have been used the concepts as early as the late 1960s; David Parnas's classic papers in the early 1970s highlighted the trend in layers and modularization in software. Today, we see a similar message in the use of object-oriented type hierarchies for reuseable software.

The essential idea is simple: If the functionality is divided into layers numbered from 0 to N-1 (0 is the lowest layer), then the system's functionality that is implemented at layer i depends only upon the interface presented by the layer i-1. In particular, the designer of layer i does not know the implementation of the layer i-1 machine (and does not even know the nature of the interface of machines at layers below i-1). In the open systems jargon, the individual layers are called *platforms* for the higher layer. That is, a platform provides a foundation on which some new layer can be built.

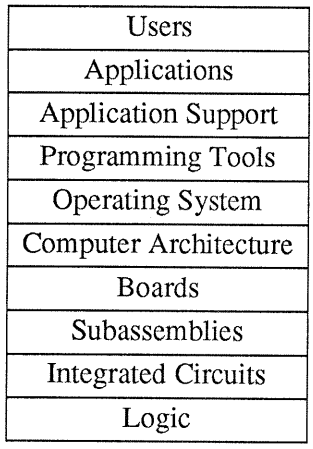

Figure 6: Layered Computer Architectures

Such an architectural approach prevents designers from taking advantage of implementations, particularly of functionality that is logically quite remote. As a result, the interfaces can be publicized -- made *open* -- and the overall system can be built by different, cooperating organizations. The layering also makes it unnecessary for each designer to understand every aspect of the machine; the designer of layer i need only be concerned about the facilities of layer i-1 and no other interfaces or implementation details.

Layered architecture can be the enemy of performance; layered implementations tend to be slow in cases where a high layer implementation makes repeated use of a function that is implemented at a relatively low level in the hierarchy. For example, a file system ordinarily performs block input/output operations on a rotating medium; if an application program is provided with a byte stream interface to the information, it may copy a block one byte at a time rather than as a bulk data transfer.

The technical challenge of layered machine architects is to understand how the layers will be used, and to partition the functionality across the layers so that implementations will be simple and efficient.

> An *open system* is one in which the components and their composition are specified in a non-proprietary environment, enabling competing organizations to use these standard components to build competitive systems.

A system, in this context, means any combination of software and hardware components. For example, a computer, an operating system, a database management system, an accounting system, or a network of computers are each examples of a system.

> A system can be *specified* if there is some precise technical description of the interface to the system.

Examples of specified systems include static RAM chips, the PC AT bus, IEEE P1003.1 ("POSIX") operating system interface, the ANSI standard for the C programming language, the SQL interface to a relational database, and X windows.

Ordinarily, one would think of open systems as being systems for which there is some standard specification for its interface. Standards are controlled by a public body, e.g., the International Standards Organization (ISO) or the American National Standards Institute (ANSI). The standard is established and modified by the consensus of all parties interested in the definition of the standard. However, there are other open specifications in wide use -- ones controlled by a specific vendor. For example the AT bus is an IBM specification, but it is used in many different personal computers.

One often hears or reads about *de facto* standards, referring to products that dominate some market. However, there may be no specification, not to mention one that is public, for *de facto* standard products. Thus we would not include products that establish or meet a *de facto* standard as an open system in our definition. Two examples of such a product are the IBM 3270 synchronous terminal and the DEC VT 100 asynchronous terminal (ANSI has adopted a standard, X3.64, which is extremely close to the behavior of the DEC terminal).

Thus open systems are composed of parts that have a publically-specified interface. This encourages the use of standard components within a (sub)system, while the system may itself be proprietary. The manufacturers of the parts can compete, since the specifications are open; thus the system integrator has multiple sources for parts.

There are several notable computer systems that use open system architectures. One of the early market successes was the Apple II personal computer (followed by the IBM PC). The success of these machines can be credited in part to the fact that other vendors could add components to the base machines in a competitive marketplace. While Apple eventually closed the II, IBM stayed with the strategy for several years, becoming more conservative in their position with the PS/2 machines. Apple has gone back toward an open architecture with the Apple Macintosh II family.

The DEC/Intel/Xerox Ethernet is another example of an open system, in this case for a local area network. The Ethernet spec started as a commercial open specification under the control of specific vendors, but has since become a public standard specification by virtue of the IEEE 802.3 specification.

There are few computer systems built today that do not use open systems approaches at some level of the architecture, be it only at the level of memory or integrated circuits. Conversely, there are no computers that employ only open system components: *Systems are neither totally open nor totally closed.*

Open systems are a logical conclusion to layered architectures -- software and hardware -- in a market where commodity components are produced. However, their popularity is due less to technical elegance than to their fundamental importance to end users. If a system has certain aspects of an open system, then it can be built from parts supplied in a competitive environment. This tends to decrease costs while increasing performance and function. It is also healthy for all vendors (except for those that control a market). Today's computer system marketplace demands an open system approaches. A product must comply with the generally-agreed upon architecture to be considered to be an instance of an open system. *This does not mean that an open systems architecture compliant product has a full implementation of all open system components specified in the architecture configured into each product!* It does mean that the parts of the product that functionally overlap with the architecture specification must comply with that specification.

It is easy to see how a networked workstation node complies with an open system architectures, since most of the architecture specifications used in the system comply with some open interface. However, suppose that the open system architecture for workstations addresses storage only for text and numeric data types, but a vendor intends to create a compliant workstation that supports extended media types such as voice and graphics. When the product is designed, the portions of the product that do not relate to the open systems architecture can be designed along an arbitrary set of guidelines, see Figure 7.

As the product emerges and as standards become more inclusive, then it is in the best interest of the vendor to employ designs that have highest probability of becoming open, and to actively encourage the adaptation of those approaches in the open systems arena.

A similar argument applies to products which use a computer as an *embedded system.* Much of the discussion of open systems architecture is devoted to user interfaces (integration), the operating system platform (portability), and network communication (interoperability). An embedded system may have no user interface, may not use an operating system, and may not exist in a network environment. Open system architectures also specify fundamental aspects of software, related to coding practices and style, device transparency, and internationalization. They also address the software development environment. Embedded systems should comply with these aspects of open systems for portability across embedded microprocessor engines and low-level device interfaces. Such an approach to embedded system design enables the controller part of the system to be flexible, and to grow into a systems and network discipline.

2. THE FOUNDATION FOR OPEN SYSTEMS

Open system and networks are of concern to hardware and software suppliers as well as consumers. The concerns of the two camps overlap, but are distinct. For any given system component, there is a *supply-side* perspective and a *demand-side* perspective (see Figure 8).

The supplier is concerned about the size of the market for a product, and the nature of the competition within the market. If a small number of suppliers dominate the market, then there is little incentive

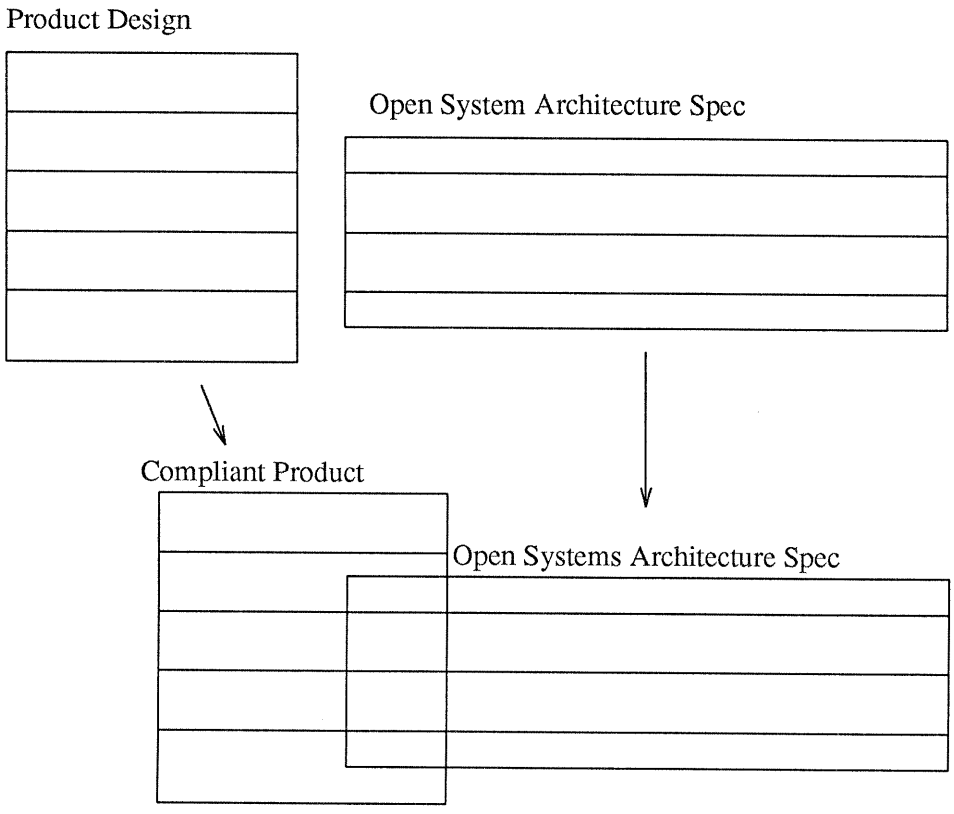

Figure 7: Architectural Compliance

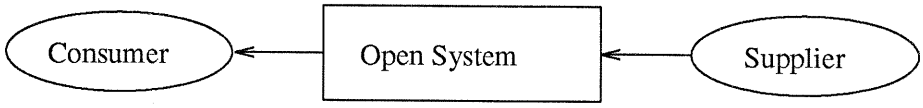

Figure 8: The Supply-Side versus the Demand-Side

for the suppliers to subscribe to an open systems philosophy. And new suppliers will not be drawn to the market unless the market size grows or the supplier creates a competitive advantage for creating products to sell into the market. In the case of open systems, the market is increasing rapidly.

Computer system products are extremely complex; each customer system is a synthesis of a spectrum of hardware and software components. The supplier into the market must be ensured that the individual component that it creates can be used with complementary components manufactured by other suppliers. Interoperability is a requirement for a successful component for these complex systems.

Interoperability focuses on peer level operation of a component within some layer of a complex system. Since contemporary computer systems have many layers, it is important that a specific collection of components that implement a level provide a well-defined interface to the client portion of the system at the next higher layer of the architecture. That is, an open system supplier must interoperate with peer components (possibly supplied by other suppliers) to provide a portable environment for the next higher layered portion of the system.

For example, an integrated chip supplier will be required to manufacture chips that can be used with peer level chips created by other suppliers. The resulting collection of chips creates a hardware platform which will support an operating systems platform.

On the demand-side of the open system are the consumers of components. A consumer may use open systems as an environment for synthesizing a collection of components to manufacture a platform ("systems integration"), or as a foundation on which to build a higher level platform ("users"). System integration consumers are concerned with interoperability and integration. Users are most concerned with the portability interface provided by the platform, (so they can achieve supplier-independence of the higher level platform that they will create).

Since consumers are obtaining components from suppliers, the consumer is concerned with the number of alternative sources that meet a specific set of interface requirements.

Notice that in the computer systems marketplace, most entities are a consumer of one set of components and a supplier of another set. It is possible to view the component being created as shown in Figure 9. Each system or system component is constructed from custom components created by the entity, commodity components, and differentiated components obtained from another entity. As the system is designed, there are always some set of components that the entity must decide if it is more profitable to build them or buy them.

The goal of the supplier to the entity is to create components that will reduce the amount of the end product that is built, and increase the amount that is purchased. From the supply-side, there are different strategies to pursue, depending upon the strengths of the supplier. For example, if the supplier has an advantage in manufacturing or sales channels, then it might concentrate on the market that supplies "commodity" components (more discussion on commodities below). However, if the supplier has a technology that will allow it to produce components at a price-performance advantage, then it should concentrate

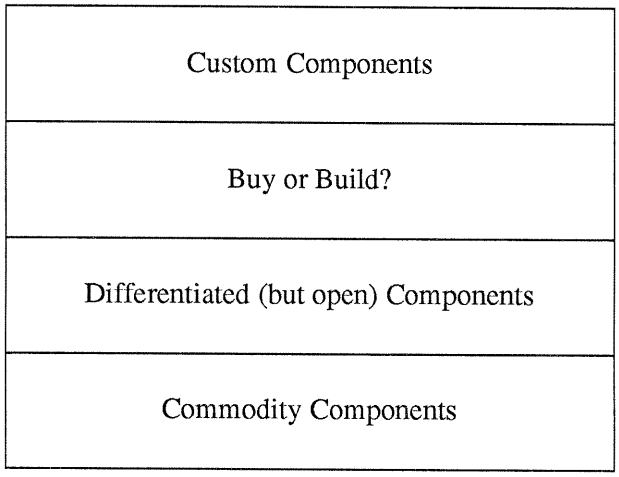

Figure 9: Build versus Buy Decisions

on "differentiated" components.

The goal of the consumer (in this part of the discussion) is to create products that maximize the technological advantage owned by the consumer. Thus, custom components are assumed to be the value added by the consumer when it synthesizes open components with its contribution. The buy versus build decision for the consumer is a balance determined by the suppliers, and the particular technology that the consumer controls. When integrated circuits were the critical technology for computers, manufacturers tended to design and implement their own integrated circuits since they could gain a competitive advantage by building a superior integrated circuit. As the technology matured, any advantage began to disappear as various implementations converged on similar price-performance curves; most manufacturers could preserve a competitive advantage by employing many standard integrated circuits with a few proprictary integrated circuits (in some cases with no proprietary circuits). The manufacturer's subassemblies or components were proprietary because the combination of the standardized components was unique.

In today's low-end hardware market, personal computers, workstations, and even inexpensive multiprocessors utilize standard subassemblies mixed with a few custom subassemblies, e.g., consider the various UNIX workstation architectures (680x0 and RISC CPUs), the IBM PC family architecture, the spectrum of shared memory multiprocessor architectures (Encore, Sequent, and others). Computer boards and buses have become standard for a large number of computer architectures.

We say that many integrated circuits have become a *commodity* rather than a technology-intensive part of the computer system.[2] A component is a candidate to become a commodity when the manufacturing costs of the component can be made economical through mass production, and/or when it is not economically effective for each system to incorporate its own custom version of the component. A producer cannot justify the cost of manufacture of the component unless he can produce high-performance components near the minimum cost in larger enough numbers to make a profit. Thus, commodity components have little to differentiate among parts provided by different suppliers. (Our notion of commodity

(2) There is still fierce competition among integrated circuit designers in cases where state-of-the-art technology is embedded in the integrated circuit, e.g., as in RISC chips or special purpose video or floating point chips.

does not completely agree with the notion used in the investment market since a true market commodity has no brand name associated with it, since there is *no* differentiation among components provided by different suppliers.)

As buses become standardized, board subassemblies tend to be treated increasingly as commodities. (This has occurred in the past for various standard buses, e.g., for the S-100 bus, the Intel Multibus, the IBM PC and PC AT buses, and the Motorola VME bus.) At the next higher level, whole single-board computers and full multi-board computers have also become commodities, e.g., the IBM PC and its clones.

A similar situation is occurring in software: While it has been effective to employ proprietary operating systems in the past, the operating system kernel is becoming less of a factor in the overall price-performance characteristics of the complete commercial computing system compared to the importance of the application software.[3] There is a clear trend towards focusing the proprietary effort on application software rather than on the operating system. As a result, the entrepreneur can maximize profit by building specialized components that are technology-intensive rather than commodities. Of course, the vendor that controls a market has little incentive to use commodity-like components if he is still enjoying a profit from his proprietary components.

3. THE EVOLUTION TO OPEN SYSTEMS

The open systems market is ultimately defined by corporate and government computer system requirements. Open systems technology became important near the end of the 1980s due to the end user requirements on computer systems as molded by the evolution in hardware and software. Open systems success depends upon agreement among vendors in the marketplace, and this leads to the need for effective standards. In this section we review factors that have intensified the need for open systems in contemporary computer systems. Corporate and business computing rose to prominence in the 1960s. In this decade, government and industrial organizations began to employ computer systems as a mechanism for storing the organization's information, and for making important organizational decisions based on that information.

As the tools for managing information increased, the amount of information that the organizations found to be useful also increased (usually at a faster rate than the technology). This information age stimulated the emergence of an important industry -- computer manufacturers -- and a major new unit within each user organization -- the Management Information Systems (MIS) departments.

In the first generation, the state-of-the-art hardware technology would support *batch processing systems* (Figure 10). Batch processing systems were capital-intensive purchases. For the organization to be able to manage the cost, and to ensure that the equipment provided equitable service to the various parts of the organization, the MIS center was created.

Once the batch processing systems were installed, it was still necessary to customize their operation through (sub)organization-specific application programs. However, few suborganization were in any position to construct these programs, so just as the hardware was managed by a centralized organization, so was the corporate software expertise (programmers).

During this first generation, the MIS center was the only corporate entity capable of obtaining, maintaining, and programming computer systems. The computer vendors were in a position to convince the MIS center that their particular product was the best, then the entire MIS center -- programmers and machines -- became "captive" to that vendor's proprietary systems.

---

(3) There is a currently a counter trend in the research community that focuses on specialized high-performance computer systems.

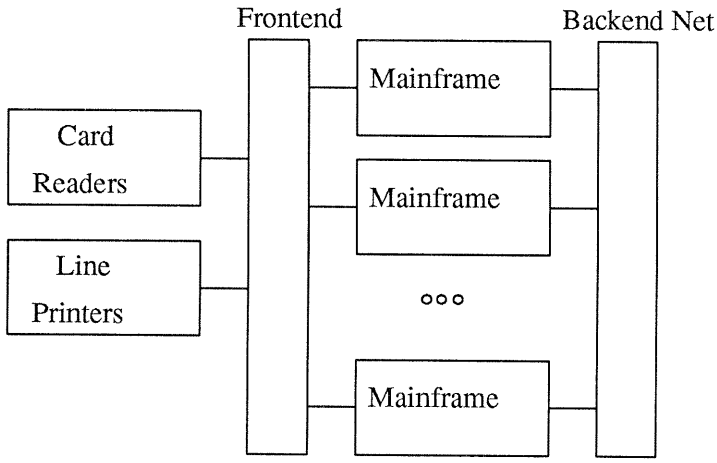

Figure 10: Batch Processing Systems

The second generation of hardware evolution -- roughly during the 1970s -- saw a trend begin to occur. First, system technology began to support the *timeshared computing* model of operation (see Figure 11). This enabled terminals to be physically placed in suborganization locations (and in suborganization budgets).

Timeshared computing systems allowed the corporation to use the MIS center for centralized expertise, control, and administration of the most capital-intensive part of the computational environment (the mainframe computers). But the end user organizations established a foothold in choosing part of the computational environment to suit their particular needs as opposed to the global needs of the corporation. The rationale was that this sort of local optimization on equipment selection was also a global

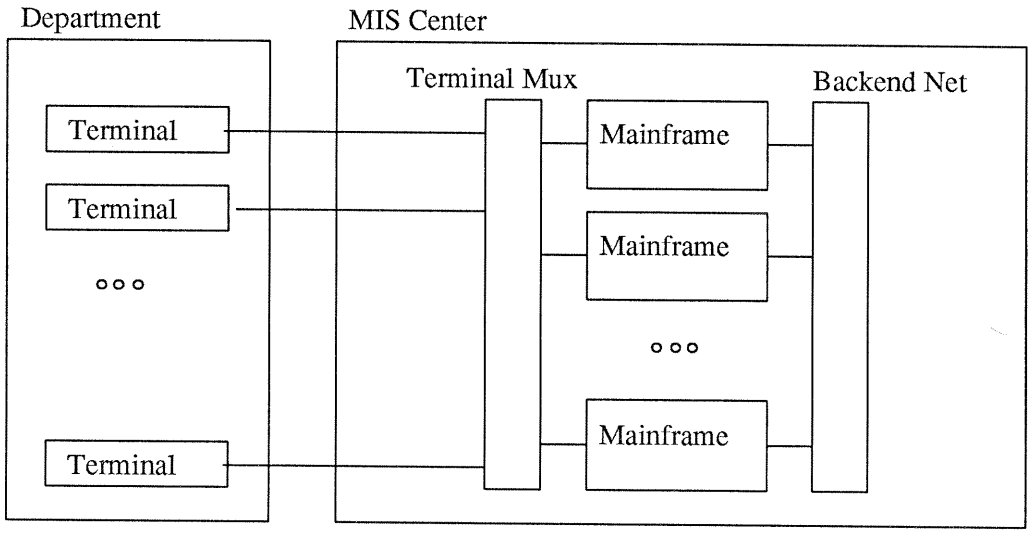

Figure 11: Timesharing Systems

optimization since it allowed the individual suborganizations to be obtain equipment that best met their individual needs.

A few aspects of an open system environment began to appear: the user interface devices could be manufactured by competing suppliers, although they had to be able to connect to the closed timesharing system. Interoperability at the terminal-mainframe level became an important issue. Not all vendors chose to allow the terminal-mainframe interface to be open; some manufacturers had proprietary character sets or line protocols. This made it difficult for the suborganization to purchase terminals from any vendor other than the one that provided the services to the MIS center.

Terminal differences and departmental-specific application requirements began to encourage the decentralization of the corporate programming resource. These suborganizations began to have sufficiently specific software requirements that it was occasionally cost-effective to have a programmer dedicated to the needs of the suborganization.

In the early 1980s, the third generation of hardware emerged (see Figure 12). There are two important hardware evolutionary steps in this generation: the introduction and proliferation of *personal computers* (PCs), and the introduction and proliferation of *departmental computers*.

Prior to 1980, the personal computer was a medium for hobbyists to enjoy programming and game-playing at a modest investment. Personal computers could be purchased for less than one thousand dollars, even though they were limited by memory, devices, and software.

In 1980, IBM announced the PC personal computer. This machine and its clones and competitors quickly penetrated the commercial marketplace as viable tools for decision support, word processing, and other computer applications. Prior to that time, most corporate computational support was provided by a centralized computer system administered by the corporate management information systems department. The PC enabled frustrated users to purchase computational power and to use it as if it were another office machine (such as a copier or typewriter).

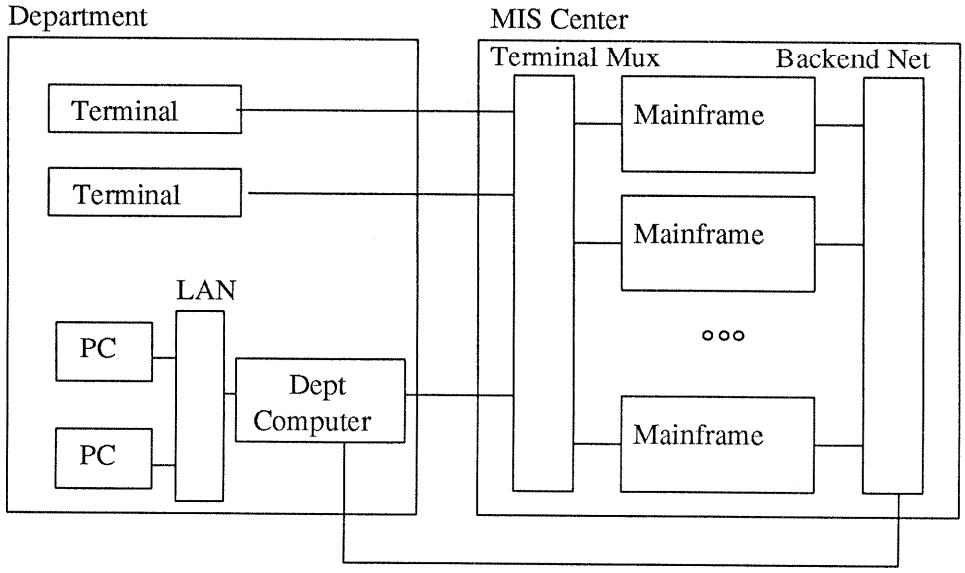

Figure 12: The Departmental Computer

As PCs became more prevalent, they began to physically displace terminals; this led to the introduction of terminal emulators which would enable the PC to be used for local computation as well as connection with the MIS mainframe machine.

The departmental computer became a cost-effective computation engine with the growth of the capability of microprocessor chips. A department found that it could begin to manage its information across individual users (each with a PC or a terminal) by creating a department computing domain of the same pattern as the corporate MIS center. In order to ensure that the local computational environment could still use the corporate information, the departmental computer provided connectivity to the MIS mainframes.

This phase of the hardware evolution is the first major sign of a revolution to the batch processing organization. The suborganization now begins to take on the role of a small MIS center, requiring its own maintenance, administration, training, custom programming, and acquisition. With acquisition policies varying from suborganization-to-suborganization, the requirement for interoperability became mandatory, and open systems became a major factor in accomplishing corporate computing goals. Within the department, multiple vendors provide PCs and departmental computers. Each of these computing platforms are shared by various users within the suborganization, running a set of departmental-specific application programs. Portability and integration become major issues facing the buyer within the suborganization.

In the late 1980s and the 1990s, we see the emergence of the fourth hardware generation -- *distributed systems* (see Figure 13). As PC applications proliferated, two constraints began to dictate PC-based computing: the local computational and display power of the PC, and the inability to share information loaded in the PC with other PC users.

The first constraint has gradually encouraged the technical marketplace to provide increasingly more powerful personal computers to a large market segment. Today, personal computers (and scientific workstations) incorporate 32-bit processors roughly equivalent to mainframe computers of ten years ago. These personal machines have also grown more sophisticated in the devices and memory which they can support at low prices: personal computer hardware has evolved into a commodity business in the last ten

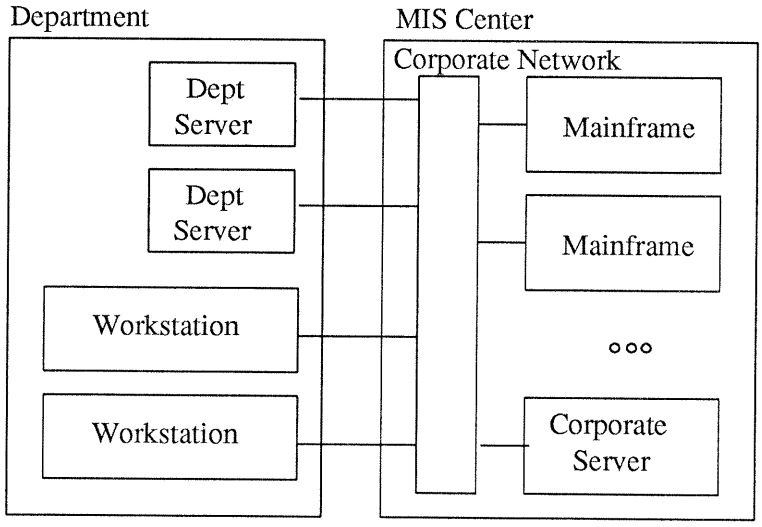

Figure 13: The Corporate Network

years.

Distributed systems completely change the role of the MIS center from the 1970s. Now, the MIS center is responsible for providing a mechanism for interconnecting decentralized computing resources. The MIS center maintains a few large mainframes that are the repository for organization-wide information, and for managing other resources that are shared across the organization. Computation is performed in the decentralized locations. Such an environment has evolved into a multivendor collection of computers, because of the evolutionary steps and because this is a cost-effective model for using computers to meet the needs of diverse suborganizations.

Open systems are mandatory for the success of cost-effective distributed systems. Interoperability can only be achieved in this environment through open systems approaches. The corporation requires portability to provide flexibility in vendor selection within suborganizations. Integration is required to make the training and use problems in such environments tractable. Hardware evolution is not generally possible without a corresponding evolution in the software. Software technology has tracked the four phases of evolution described above.

During the first and second hardware generations, open systems concepts were not generally an issue. The third generation hardware platforms highlighted issues concerned with interoperability, integration, and portability.

The communications network and the operating system have been the first components to evolve as a result of the requirement for computing in an open systems environment.

The activity in communication networks has stimulated considerable activity among all of the computer vendors. Two communication models have survived the 1980s and are major factors in interoperability: the International Standards Organization (ISO) *Open Systems Interconnect* (OSI) architecture, and the IBM *Systems Network Architecture* (SNA). ISO OSI is clearly in the open systems domain and IBM SNA is at least partially open even though it is a proprietary approach.

In the operating system arena, a single open system approach interface is emerging (although there is considerable effort to accommodate a small number of proprietary operating systems). The IEEE P1003 *POSIX* operating system interface has emerged as the most widely-accepted definition of open operating system interface. POSIX has evolved from AT&T UNIX, although it is controlled by an open system process. However, because of the wide-spread use of the Microsoft MS-DOS (IBM PC-DOS) operating system for IBM PC-compatible platforms, DOS is also a major factor in open system considerations. (That is, the DOS interface is not controlled by an open requirements process, but the interface specification has been made generally available, and it has been used in so many applications that it has created a portable environment across many vendors' hardware platforms. The effect is as if DOS were an open system specification from the consumer's point-of-view.)

Over the last few decades, software has failed to keep up with the hardware evolution. Today's software systems are only marginally more adept at handling multiple-machine computations than they were in 1980. While the hardware technology has raced ahead with fast, inexpensive machines with relatively high-speed interconnection devices, there is a strong limitation on the ability to utilize the hardware in an efficient manner. This is the focus of distributed systems, and an important factor in today's software crisis.

In contemporary computer systems, the pace of hardware evolution has so far exceeded that of software, particularly applications, that there is increasing risk associated with creating innovative hardware. Consider a hardware manufacturer that controls only a small fraction of the total installed computer base: if that supplier were to market a machine that were too radical, then the only software available for the machine would be that which was written explicitly for the machine. Since the

manufacturer controls only a small fraction of the market, it is unlikely that many independent software organizations will produce software that is well-tailored to the manufacturer's unique hardware. The product can only succeed by generating a special class of programmers (the approach used for exotic computer architectures such as a connection machine or a SIMD machine), or by providing some uniform interface to the hardware, preferably as a software layer. This is one reason that the X/Open consortium of hardware suppliers was formed. The consortium members are competitors, and the X/Open Common Application Environment is a collection of software standards -- operating systems, operating systems extensions, and networks -- that the members support with their hardware. The Common Application Environment provides the members with the freedom to produce innovative hardware products while preserving a market in which common application software can by used.

4. CONCLUSIONS

The march toward open systems is inevitable, with ever-increasing pressure of the requirements of the end user organizations. In today's economy, it is difficult for a corporation to rely on the proprietary products of a single manufacturer. The technology has evolved to the point that multivendor installations are plausible and cost-effective, strategically enabling the customer to select components from various vendors and to integrate them into a corporate computing environment.

The manufacturers understand and accept the evolution toward open systems. While those manufacturers that dominate a particular niche in the corporate computing market will be less enthusiastic about open systems than those who do not, they have began to evolve toward open systems.

How can manufacturers, ISVs, VARs, system integrators, and end users prepare for open systems? First, it is necessary for the players to understand the big picture in the technology, not just one view of it. Each of portability, interoperability, and integration are an important part of the big picture, but none individually provide the full perspective. To address open systems effectively, one must be knowledgeable about all of the views.

We see an exciting time in business and technology in the next decade, stimulated by the continuing evolution to open systems. Of course, this provides a major new opportunity for realignment of technology, products, and organizations in corporate computing. The strategic planners of supply side and demand side organizations will need to address a wider scope of issues than they have in the past, and the techologists will have to learn that the leverage in effective products will be in the non-commodity parts of the system -- application support and vertical applications. The most successful software of the 90s is more likely to be application support packages rather than horizontal application software such as publishing and spreadsheet packages.