NASA 1111 [illegible]

NASA Technical Memorandum 86438 NASA-TM-86438 19850022397

A New Taxonomy for Distributed Computer Systems Based Upon Operating System Structure

Edwin C. Foudriat

June 1985

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

A New Taxonomy for Distributed Computer Systems Based Upon Operating System Structure

by

Dr. Edwin C. Foudriat
Langley Research Center
Hampton, Va 23665

Abstract

The paper considers characteristics of the resource structure found in the operating system as a mechanism for classifying distributed computer systems. Since the operating system resources, themselves, are too diversified to provide a consistent classification, the author examines the structure upon which resources are built and shared. O/S resources must run concurrently and determinately. The underlying structure to accomplish this is the indivisibility or atomic activity provided by the O/S kernel and data structure. The location and control character of this indivisibility provides the taxonomy for separating uniprocessors, computer networks, network computers (fully distributed processing systems or decentralized computers) and algorithm and/or data-control multiprocessors.

The taxonomy is important because it divides machines into a classification that is relevant or important to the client (user of the machine) and not the hardware architect. It also defines the character of the kernel O/S structure needed for future computer systems. For example, based upon the kernel analysis, it is apparent that computer networks are easy to develop from uniprocessors in a value-added fashion, but that true network computers are not yet available because development by extending computer network operating systems is not practical. The paper also discusses in detail what constitutes an operating system for a fully distributed processor.

N95-30209 #

1 Introduction

There have been many promises made about the capabilities of distributed computer systems--new features like increased computing power, greater reliability, more flexible user environment--which will usher in a new computing era. Papers and proposals, espousing large numbers of computers operating in parallel doing millions of operations per second (MIPS), can be found throughout the literature [1, 2]. The low-cost microprocessor and local area network technology have stimulated many configuration and systems proposals.

More recently, a degree of realism has been injected on this euphoric scene. For example, Allchin and McKendry state [3] "It is not clear, however, that current technology is able to realize these advantages. Without advances in methodologies for constructing distributed systems, we are faced with a situation in which we are likely to see less, not more, improvements in these areas." Jensen echos a similar theme in describing and justifying the ARCHONS project [4].

The most popular form of distributed system found today is largely an evolution or a simple adaptation of the uniprocessor technology (centralized computer) with a value-added network system. To quote Jensen [4], "A network normally is supplied with a so-called "network operating system" which tends to be simply the collection of network server utilities. Historically, it has been constrained to be a guest of the local operating system." He then supports McKendry's arguments with " Evolution is generally appropriate as a primary mode of computer (and other) system development, but it should be performed with much careful thought. Almost all work on "distributed" systems in general, and "distributed"/network operating systems in particular, has been evolutionary to an extreme...most of the resource management concepts have been simple adaptations of centralized ones, burdened by inappropriate and even counter-productive artifacts."

In this paper we examine the structure of distributed computer systems based upon the features that can be supported by the operating system. Since almost all operating systems differ, we look for the commonality which will allow classification--the character, scope and location of the structure which supports the construction of the operating system resources. This, in turn, allows distinct and reasonable separability among distributed computer systems. It also permits the establishment of classes as seen by the operating system client or user. Since the operating system provides the shared resources which the client uses to build useful problem solving structures on the computer system, the taxonomy addresses fundamental issues of interest to the client by dealing with the basic character of the computer resources.

The taxonomy also addresses the problems concerning evolution and better methodology noted above. It demonstrates why new methodologies are needed, and why the present evolutionary nature of distributed systems supports mostly loosely coupled networks. To accomplish these, the paper first reviews present computer taxonomies. Based upon one taxonomy, it considers operating system issues like communication activity, latency of information, resource management, etc., illustrating problems of each as a mechanism for classification. The paper then introduces the concept of indivisibility (atomic structure) and where and how it is implemented for use by the operating system. Based upon indivisible features, distributed systems are classified into four categories: uniprocessors, computer networks, network computers, and algorithmic-data control processors. The paper concludes with some general observations resulting from the taxonomy.

2 Taxonomies for Distributed Systems

Various taxonomies exist for classifying distributed computers systems; many are based upon topology, switching capability, etc.,[5]. Wittie examines network structure to look at features like delay, connectivity, critical failure points, etc., [6]. Others have classified distributed computers according to the layers, like the ISO standard seven layer model [7]. Alternative to layering, one can examine distributed computers based upon global issues like naming, error control, etc. [8]. In the future, when concepts for these global qualities are better established, comparision between methods for mechanization based upon issues and system performance will be necessary [9].

Flynn [10], in his classification of computer architectures, identified single- and multiple-instruction-stream, multiple-data-stream (SIMD and MIMD, respectively) computers as those in which single or multiple instruction sequences operate simultaneously on different sets of data. A network of computers is an MIMD computer built from independent, asynchronously executing, coupled processing elements. This classification correlates more closely with the type of problem which the distributed system can handle, so we will look more closely at MIMD based taxonomies.

A review of some other computer-communication system classification schemes is found in the IEEE Network Tutorial [11]. Classifications include functional view, designer's view, manager's view, etc. The functional view is closest to the interests of this paper; it divides systems into remote-access networks (RAN), value-added networks (VAN), and mission-oriented networks (MON). However, the discriminating and distinguishing characteristics of each are only briefly sketched.

Wittie and Van Tilborg [12] carry the functional classification of MIMD machines further by defining three categories:

1. Computer networks -- independent computers somewhat arbitrarily connected to each other. Generally, a common network communication protocol connects "each" host to a subnetwork of interhost communication facilities. Hence, the computer network embodies the characteristics of the network operating system. This category includes RAN and VAN groups, above. Examples of computer networks include ARPANET [14], CSNET [15], office automation systems [16], etc.

2. Multiprocessors -- The property that characterizes this type of computer is the inclusion of a single physical address space that can be directly accessed by all of the cpu's. Usually, all of the cpu's use the same instruction set architecture and execute the same operating system. Examples include CM* [17], X-TREE [18], FEM [19], etc.

3. Network computer -- a species of compact computer network specialized to facilitate closer cooperation among processing elements. A variety of interconnnection structures have been proposed for network computers including cubes, hypercubes, trees, rings, bus clusters, etc., in order to control this closer cooperation [6]. A cohesive operating system joins the nodes into a coordinated computing machine. The MON group, above, fits best here.

The taxonomies above, classify computers physically by the mechanization of their communications structure, primarily distance, and hence, intuitively by the speed at which they are able to communicate. This, in turn, implies their ability to coordinate their activity so distributed computers should have the following operational features: {*}

1. computer networks have local control, accessed resources, loosely coupled, node autonomy;

2. multiprocessors have global control, transparent resources, tightly coupled, no node autonomy;

3. network computers have local-global control, shared resources, closely coupled, shared autonomy.

The taxonomy above, while generally clear in the large, does not resolve many issues. First, all systems are divided based upon their ability to communicate. Since communication rates are increasing rapidly (optical rates of 2 Gbits/sec. are on the near horizon [20]), a new network system with an optical bus may really be faster than an old shared-memory multiprocessor. Second, all of the definitive adjectives above are relative. It is not clear that these features automatically accrue as a result of just closer physical coupling. Finally, some machines like CM* have shared memory, but based on their use, they exhibit properties more in keeping with computer networks. Typically, even shared memory systems allot a large segment of each memory to a local processor. When large quantities of memory must be shared between a number of computers, processing power falls off rapidly. Thus, these machines must be as careful about placement of data and programs as network systems. It becomes further clouded when

{*} Gligor [13] carries functional classification further by considering multiprocessors, multiprocessors with local memory, distributed storage, central (peripheral) networks, (decentralized) computer networks, and internetworking. This further detail does not provide additional discrimination of the operational features.

the communications control features are examined, since systems in all three categories use message passing. Hence, while these systems look different to the builder or hardware architect, they look the same or similar to the user.

Clearly then, it becomes difficult to separate MIMD systems based upon the physical structure and speed of their communication system, especially when related to the class of problems which each can solve. The environment we wish to create in the network computer is distinctly different from that of the computer network; it is characterized by a multiplicity of resources, physical distribution, unity of control, network transparency where desired, reliability through replication or similarity of resources, and component autonomy. This according to Enslow [21] is the Fully Distributed Processing System (FDPS); hence, the goal of the FDPS, the client's view of Wittie's network computer and Jensen's decentralized computer are synonomous.

3 Operating System Issues

Since the client's view of a computer is generally based upon the resources provided to him by the operating system [22], it should be interesting to examine the characteristics of distributed computers based upon their operating system. As noted by Wittie [12], "Achieving the goals of parallel computing, machine extensibility and fault-tolerance now depends more upon the availability of suitable operating systems than it does on the construction of network hardware, ..." He attributes this observation to Siework in 1975 [23]. Even though the features of the operating system were recognized to be critical to parallel computing in the mid 70's, they have still not been satifactorily addressed. Joseph, et al. [24] in 1984 notes that allowing a single user program to run distributedly on different processors requires mechanisms not supported by operating systems.

Difficult practical and theoretical decisions concerning decomposition, distribution, and synchronization of tasks; parallel applications languages; communication system protocols; scheduling and sharing of resources; concurrency control; detection and control of deadlock and recovery from hardware and software failure exist for distributed operating systems. While one could possibly separate computational systems based upon some combinations of the above factors, their distinguishing features might be more difficult to discern than the taxonomy based upon physical coupling of communication systems. To illustrate, we will examine a few of the factors in greater detail.

One separating feature might be based upon decision making techniques. Both Wittie [12] and Jensen [4] examine this issue, noting that resource allocation is difficult for a number of reasons:

1. the network computer nodes do not know where to find a particular resource;

2. the data structure for global management of resources becomes too extensive to fit upon one machine;

3. the level of communication activity to share all information would exceed the system capacity;

4. the communication delays play a substantial roll; by the time the information gets to the proper node, it probably does not reflect the true situation;

5. the failure of a network computer node must be handled in such a manner that processing can continue;

6. the decision making system is much more complex in order to compensate for factors 4 and 5.

Jensen dwells upon the sixth aspect of the problem in much greater detail. In examining resource management, he comments [4] "Unfortunately, operating systems as presently conceived are highly and inherently central in several critical aspects. Perhaps most importantly, they are based upon some very strong premise about time; e.g., that communication delays due to physical dispersal within the operating system are practically negligible with respect to the rate at which the system state changes (note that the same effect can occur on a single VLSIC/VHSIC chip). This leads to the presumption that it is possible (and even cost effective) for all processes to share as complete and coherent a view of the system as may be desired (e.g., that a single ordering of events can be established)." Jensen also eludes to point 5, the failure problem, noting that both information latency and failure are mainly considered (up to now) in the context of shared memory systems.

Decision making is separated into logical and physical factors. Jensen further states that logical decision making concepts are needed for decentralized computing, implying that this is a primary reason why "distributed" and network operating systems are so disparate. He proposes concepts like multilateral management noting that decentralization must be founded upon it; "not for instance on the more common theme of resource or functional partitioning." Hence, decisions must be based at least upon "team" effort or even more decentralized upon some form of partitioned competence or disparity of information with the activity eventually converging to a single consensus decision. "But the region of primary interest to us in the multidimensional space of logical decentralization is where each global decision is made multilaterally by a group of peers through negotiation, compromise and consensus."

While the above concepts make nice philosophy, we do not feel that all network computer operating system resource allocation must be based upon concepts like group and consensus decisions; that is, we do not feel that this activity constitutes a discriminator for taxonomy purposes. We note that such schemes may create additional problems; for example, "If you want nothing done, give it to a committee." Hence, distributed systems will make some resource decisions based upon limited, imperfect knowledge in an arbitrary and capricious manner, using the philosophy that sometimes it is better to do something, hopefully useful, than just to talk about it. Therefore, decision making while nice is not necessary for network computers. {*}

The fourth factor above, inaccurate and insufficient information, as a distinguishing difference between decentralized and centralized or network operating systems is harder to dispel. We present two arguments which illustrate that this feature is not a discriminator either. First, on the surface the situation that a central operating system can gather lots of useful and coherent information to make perfect decisions, appears to be reasonable. Practically, they don't. Optimal decisions on resource allocation are many times NP-hard and operating systems can't take the time (like peer decisions) because this would permit less useful work of the client to proceed. Hence, they attempt to make reasonable and consistent decisions to provide correctness preservation for the resource and some useful ordering of events that will allow the clients to proceed and the O/S to recover in case of failure or error. Furthermore, optimization is goal related. Changing the goals may change completely the decision making scheme and, hence, the information required by the operating system. Since O/S functionality is usually decided long before the complete utility of the computer, optimum goals are not known.

Second, with regard to the timeliness of information, both Jensen and Wittie dwell upon the communication delay in the network as being a large factor in differentiating distributed operating systems from their centralized or network counterparts. (Jensen, however, notes correctly that delay may be a factor in closely coupled systems like a VHSIC chip.) To be precise, information delay and lack of singular global ordering occur in all computer systems, including uniprocessors. For example, disc access and resultant DMA to main memory takes considerable time and may occur in an almost infinite variety of ordering with respect to CPU instruction execution; the operation is completely under control of the local backplane access, priority, and structuring mechanisms. Delay that occurs during the completion of the DMA event may cause operating system problems. For example, failure may not be known to the operating system until it checks the status register. This may cause it to abort previously made decisions. <u>Delays and ordering in any modern computer cannot be completely controlled by the operating system.</u> In fact, different backplanes impose radically different timings with regard to information exchange and orderings, yet the operating system provides a completely consistent abstract view to the client. Hence, when examining the fine detail, uniprocessors exhibit many characteristics equivalent to distributed systems. In fact, what is a backplane but a communication system between distributed (albeit, special purpose) processors.

{*} We do not mean to imply by these statements that negotiated decision making is not an important and fruitful topic for research in distributed systems. We support and endorse Jensen's ARCHONS program; we state only that the concept does not discriminate between classes of distributed systems.

Although the primary goal of the operating system should be resource management, we disagree with the previous referenced work [4, 12] that decentralized systems differ from centralized ones because of the inherent communications delays or the logical features of decision making. In fact, the primary distinction between computer network and network computer operating systems is not any one or a combination of factors listed or discussed previously. However, we do agree with the conclusion that they are distinct and with Jensen's [4] and McKendry's [3] conclusion that evolutionary development from computer network to network computer is inappropriate and ill advised. In fact, we strengthen that argument to show that it is not practical and probably not feasible.

4 Indivisibility Concept In Operating Systems.

As noted previously all operating systems must cope with the ordering and delay problems. These problems may exist because the resource is hardware related and not under control of the operating system or software related, occurring because of the structure created and used by the operating system. This problem is related to concurrent processing, in that the CPU should do other useful work until an event takes place. Coffman and Denning [22], in discussing operating system theory, note:

1. the desire for efficient utilization of the equipment leads to the widespread use of concurrency among the central machine and its peripheral devices;

2. the desire to share information and communicate among the existing programs; and

3. operating systems are thought of as a control program that coordinates various concurrent activity.

To implement concurrency, the concept of processes or tasks have been defined. In parallel processing, more than one process may be observed between initiation and termination stage, and even in a uniprocessor, more than one program may be progressing simultaneously. For example, one program may be utilizing the CPU, a second suspended awaiting an event, and a third running on an I/O channel. Hence, all modern computers operate concurrently. To differentiate between computer systems, we examine the mechanisms they use to control that concurrency.

A process to run correctly must be determinate [22]; i.e., its termination state must be a result of its initial state and inputs, not a result of the speed or schedule under which it runs. In order to accomplish determinacy, processes must employ indivisibility or atomicity in its critical activities. Indivisibility, in the context of the operating system, causes services to perform determinately. For example, to guarantee correctness, an operating system handler may block (synchronize) any entry which rewrites the device memory control structure (memory mapped I/O) until after the scheduled interrupt has taken place and may then place the processor in an uninterruptable mode until the handler completes some critical activity, like a check for errors, resetting the I/O controller variables, etc. A software service, like a queue or stack, may have data and code controlled through mutual exclusion or synchronization structures to guarantee two clients are not given the same data space.

Fundamentally, each O/S resource may require a unique form of indivisibility. While the mechanization of indivisibility varies widely among architectures and O/S implementation, its scope, the location where its support is found and who controls progress is identifiable. First, indivisibility is found in the hardware-firmware-software that supports or underlies the operating system, usually called the kernel. Also, it can be found as data structure in the operating system itself. In addition to scope, thread-of-control, the control characteristic of the indivisibility implemented by the O/S and its kernel, is important. We will use the scope and control character of this indivisibility to develop a new taxonomy for distributed computer systems and show that it is a useful tool in analyzing the capabilities exhibited by various computer systems.

Before we discuss the indivisibility features incorporated in operating systems, we need to consider the manner in which resources can be viewed. We adopt Allchin's concept of abstract and concrete [25] (closely related to Jensen's logical and physical [4]). Resources maintain two views: abstract, the view presented to the client for his request/response operations; and, concrete, the view more closely related to the need to maintain the resource (more closely related to the physical). The abstract view should provide a consistent, useful service to the client, and the concrete view should control the internal state of the resource to maintain consistency and recover from failures. If a failure should occur, the client should understand the cause of the failure (if related to his activity) and the resultant state of the resource. Obviously, certain resources can present differing views to differing classes of clients and be combined to provide a higher level of service to clients.

5 Structures for Distributed Computers Related to Their Operating Systems

In Figure 1, we show the four classes of distributed computer systems we intend to consider. All four of the systems show concurrency and indivisibility (atomic activity) and hence, should be considered for inclusion in our taxonomy.

5.1 Uniprocessor

Figure 1a) illustrates the structure of the uniprocessor; while not strictly considered a distributed system, it provides many features which clients desire a computer to possess in present and future machinery. For example, multi-user capability is supplied by an O/S that is capable of regulating and sharing its resources so that each user preceives that he has full control of the machine.

Let us examine the uniprocessor indivisibility characteristics. First, at the physical level, architecture may support DMA, test and set, interrupts, hardware exceptions, software traps, virtual memory traps, etc. At the kernel level, support may exist for concurrent process control including context switching, critical section control, synchronization with P & V operations, etc. While the details of the indivisible structure are not important, the scope and thread-of-control characteristics provided for resource sharing are related to a single CPU only. To state differently, a single processor deals with indivisibility structures local to its environment. For example, only one processor responds to interrupts. At the abstract resource level, the uniprocessor provides for storage of information in the form of files, starting and stopping tasks, etc. with certain characteristics based upon indivisible data structures (e.g. file directory) created, processed and stored at the operating system level. Thread-of-control to deal with these resource structures is based in a single processor. In both cases the system indivisibility is related to a single processor activity only. Based upon this finding, we classify this computer as a uniprocessor.

5.2 Computer Network

Figure 1b) illustrates the computer network system where the local O/S is augmented with a network in a valued-added fashion, as noted above. These combine to form the basic network operating system. Here, the client preceives himself to be connected to a single node but with the capability to access other nodes.

Let us examine the computer network indivisibility characteristics. Since resources like storage, hardware interrupts, etc. are tied to a particular machine, local or single processor indivisibility exists similar to that of the uniprocessor. The network firmware provides additional indivisibility through the transmittal or receipt of information. While the nature of this information structure varies, the physical structure is usually based upon a block of information (I/Q buffers) and controlled similar to other I/O channel information at the local level. No additional structure is added to the kernel to support the communication system.

At the abstract resource level, the O/S provides a communications resource in addition to all the resources provided by it as a uniprocessor. These resources vary but recently standardization of protocol structure [26] is being implemented to enable compatible communication across a wide variety of machines. Protocols deal with the message structure of the communications system.

The communications system may contain a number of interesting thread-of-control features some of which provide indivisibility across nodes. The characteristics of the routing system are particularly interesting. In some networks routing is controlled locally with some information for routing decisions supplied either locally or globally. In others, routing is strictly global e.g., controller-in-charge (IEEE 488) [27]. The controller-in-charge is an example of non-local indivisibility structure in that another CPU has access to and control of information structures which influence the local machines progress or scheduling of its resources. Globally supplied routing tables may or may not exhibit non-local indivisibility depending upon how the information supplied globally is synchronized with its local use.

Another interesting communications system structure handles the acceptance and flow between machines. The acknowledge protocol has non-local indivisibility; i.e., the acknowledge processing on one CPU controls the utilization of resources (buffers) on another CPU. Mechanisms to enable choking control or rerouting when buffers or channels get overloaded are related examples [33].

Another example involving more than one computer and structures other than the communication thread-of-control is the IPC structure for remote procedure call (RPC) [29]. Here, RPC basically causes processing to begin on another machine. Philosophically, locus of control for the program progression is transferred to the new machine; it is not normally shared between machines. Also, data sharing is implemented on a call-by-value and call-by-value-return because there is usually no way for the operating system to control the indivisibility necessary to organize call-by-reference variable control. Hence, we would classify the RPC as a local indivisible structure. Using a surrogate program [30] to implement the control further divides the processing into local controls on each machine. However, the communication structure may still maintain non-local indivisibility with its acknowledge and/or time-out structure.

Finally, consider the example of a distributed database [31]. Here atomic activity based upon transaction indivisibility exists. Data structures on different machines are clearly involved when locks are obtained and maintained on all data accessed in the transaction; two phase commit [31] involves structure and control across machines. Hence, a distributed data base implemented by the operating system is clearly a non-local indivisible activity. However, most distributed databases are implemented at the client level [36].

With this clearer picture of what constitutes indivisibility features, we can return to consideration of the computer network. We define the computer network to have local indivisibility structures for all its resources except the communication system which has non-local indivisibility. A computer network does not have distributed O/S indivisibility structure for any activity other than the communication system.

It is precisely for this reason that network operating systems can be created in an add-on fashion. In fact, they can have their communication system indivisibility features incorporated in separate computers like the Interface Message Processor (IMP) for ARPANET [7].

5.3 Network Computer

The next category of computer system is shown in Figure 1c). Here the operating system is still local. However, it supports a client who sees the system as an FDPS [21], network computer [12] or decentralized computer [4].

Since our claim is that we can classify the computational system by the indivisibility structure in the O/S and the kernel support, let us examine the system structure for the network computer. While it would be most desirable to do so by examining existing examples, Jensen [4] comments "Presently, ARCHONS appears to be essentially alone in stressing unification at the operating system level. ... The only popular alternative to conventional centralized computers is computer networks." Joseph, et al. [24] also cites the lack of available commercial operating systems that distribute program parts. Thus, we have to examine proposed systems.

We first note that, like the previous systems, local indivisibility exists. Second, like the computer network, network control structures must be present although they may be implemented differently. The additional feature required for network computing is indivisibility control structure at the kernel and O/S level which implements generic distributed activity. Such features, proposed by McKendry and Allchin for CLOUDS [3, 25] and by Jensen for ARCHONS [4, 34], are based upon a generic atomic transaction system.

In CLOUDS the basic structure is the object. It supports many different kinds of concurrency control. It also supports atomic transaction protocol such as Begin Action, Commit, Abort, and End Action, and recovery based upon volatile and permanent storage. The kernel provides the underlying data and procedure structures to support transaction activity across the network. As a result, the network communication handling must exist at the O/S kernel level, a distinct change from the computer network operating system. Fundamentally, the object and action structure supports the construction of truly distributed information resources.

Similar structure exists in ARCHONS; "... this suggests that an atomic transaction facility for use by the O/S..." [4]. With such a facility, resource structures that are truly distributed can be implemented and provided to the clients. The clients may preceive the abstract resource as transparent, generic to the total system or as existing over a group of processors which, by virtue of their physical or logical information resources, are able to provide the service. Both CLOUDS and ARCHONS are engaged in implementing the support for distributed transactions at the kernel level.

Thus, in our taxonomy, network computers, decentralized computing and the Fully Distributed Processing Systems are synonymous and identified by the fact that their operating system kernel supports control for distributed information. Hopefully, in the future, hardware, software and firmware developers will find newer and better methods for implementing transaction and other indivisibility structure across computer system networks so that the next generation of computers can have more efficient distributed resources.

5.4 Multiprocessor -- Algorithm - Data-Coupled Computers

Figure 1d) illustrates the final computer which we have called the algorithmic or data coupled computer. This computer, like the multiprocessor in Wittie's taxonomy, has many local cpu's operating in parallel. However, by careful examination of the figure, we see that it is structurally identical to the uniprocessor since both the operating system and the client view it as a single system, albeit, constructed somewhat differently from the uniprocessor. The operating system may preceive the computer as having local support for a global operating system plus a truly global operating system controlling the resources. The client may preceive that certain processors do certain kinds of processing substantially better than others, but he still communicates with the O/S as an integrated set of resources. Note, there is no distinction yet from a uniprocessor which may have a central cpu, another cpu controlling the disk, another controlling the terminal, etc., all connected to a backplane with shared memory, memory mapped I/O, DMA, etc.

The distinction from a uniprocessor must be found in the indivisibility provided in the system. Like the uniprocessor, local indivisibility exists at each node and depending upon the type of information exchange, communication system indivisibility. In addition, algorithmic or data-control indivisible structures, based upon the operations being performed or the manner in which the data is supplied to the individual processors, is what gives the algorithmic processor its unique characteristic. For example, the CDC Star (actually an SIMD example) is a vector machine in that vector operations can be executed. By specifying vector structures (address and length) in calling a vector instruction, mutiple operations are performed (e.g. vector addition) indivisibly upon the complete data structures. Alternatively, a machine that implements correlation or neighborhood processing, like the MPP [32] or FEM [19], respectively, transfers data to its immediate neighbor processors in a prescribed fashion between operations, again, in an indivisible fashion. Note, it is this kind of indivisibility which differentiates the algorithm or data-coupled computer from the computer architectures considered previously. There are a great number and variety of algorithmic and/or data-coupled processors; in fact, the client usually asks what equation or math function does this one solve?

Hence, computers can be divided into four categories based upon the indivisible nature of the information and control that exist at the operating system level. These categories correlate closely with those the client sees in the resources which he can control and by the character of the problem he can solve most easily. They are:

1. Uniprocessors - local indivisibility

2. Computer networks - local and communication system indivisibility

3. Network computers - local, communication and distributed information indivisibility

4. Algorithmic or data-coupled computers - local, possibly communication, and algorithm and/or data-control indivisibility

6 Discussion and Observations Relating to O/S Indivisibility Computer Classification Schemes.

First, in operating systems that support a particular classification, it is usually feasible for the client to implement additional control structures in order to use the machine in a manner not supported by the operating system. For example, in a computer network, transaction based protocol can be constructed to provide a distributed database or an algorithmic processor by building proper distribution and control structures. Most research on distributed database management is of this form [35, 36]. This is certainly practical if done on a limited basis. However, since it is client built, it is difficult to integrate its control with other client-built resources, especially if they must share information. Also, in many programing systems, it is hard to build in the concurrency needed to make the system run efficiently since concurrency is under control of the operating system. In addition, if the client structure includes a new physical resource, then additional difficulty exists in integrating that resource in with the other operating system controlled resources. Finally, it is nearly impossible to integrate the client-built resource with other O/S resources.

Second, we note that the taxonomy deals only in a limited way with physical or communications hardware. For example, indivisibility structures in the computer network discussion have been related to some physical structure, e.g., the controller-in-charge protocol. However, this protocol could easily be developed using a shared memory computer assuming some form of interrupt or notification system. Alternatively, a shared memory system may be structured to perform like a computer network with control based upon message passing but without some of the formal standard header protocol. In different systems, indivisibility structures can exist in either hardware, software or some combination. To the client, different hardware structures may exhibit similar resource characteristics. While the computer hardware architect considers shared memory and networks distinctly different systems, the client may be faced with the same programing problems. This also implies that certain tasks might run like they were on a computer of a different class depending upon the structure of the O/S resources that they use.

Finally, we note that our indivisibility argument supports Jensen's observation that evolution of the O/S from the computer network to the decentralized computer is not practical. Distributed structure at the

O/S kernel level is required to support transaction and concurrency. This, in turn, requires a communication structure and other distributed system support at the kernel like naming and location mechanisms. Such a kernel structure is outlined by McKendry for CLOUDS. It seems unlikely that a present operating system kernel and data systems structure can be augmented since the network computer system is substantially different from the present network operating system structure. In order to provide decentralized computing, FDPS, or the network computer system, a totally new operating system development, like CLOUDS or ARCHONS, is required.

7 Concluding Remarks

Operating system resources define the basic characteristic that computing machines provide to their clients. While operating system resources have a great diversity; they all share the requirement for concurrency and determinacy. There latter features are provided by the indivisibility or atomicity structure provided by through the O/S kernel and its thread-of-control. By examining the scope and location of these features, we can classify multiprocessor systems into four categories:

1. Uniprocessors

2. Computer Networks

3. Network Computers

4. Algorithm and/or Data-control Computers

This classification is important since it supports the client's view of the computer system utility and not the hardware or system developer's view. It particularly emphasizes the difficulty clients have because of the inherent scope of the computer supplied resources. The author anticipates that in the future more detailed examination within each classification may better define the user's view of particular computer systems.

The taxonomy and its justification also indicate the type of O/S support structure needed for truly distributed processing systems. In this respect, the examination of the character of the indivisibility structures for the network computer show that it is not practical to construct such operating systems by extending present computer network operating systems.

Finally, further examination of indivisibility structure for each class, and especially the network computer class should point the way to new concepts in hardware, firmware, O/S kernel software and O/S languages that will provide better and more efficient structure for future computing machinery.

8 References

1. Lampson, B. W.; Paul, M.; Siegert, H. J.: Distributed System Architecture and Implementation, Springer-Verlag, Berlin, 1981

2. Elson, B. M.: "Intel Offers New Family of High-Speed Computers for Scientific Applications," Aviation Week & Space Technology, March 4, 1985, pp. 81-83

3. Allchin, J. E.; McKendry, M. S.: "Support for Objects and Actions in Clouds : Status Report," Tech. Rpt. GIT-ICS-83/11, Georgia Inst. of Tech., May 1983

4. Jensen, E. D.; Pleszkock, N.: "ArchOS: A Physically Dispersed Operating System," IEEE Distributed Processing Technical Committee Newsletter, June 1984

5. Ibid 1, Ch. 5

6. Wittie, L. D.: "Communication Structure for Large Networks of Microprocessors," IEEE Transactions on Computers, Vol. C-30, No. 4, April 1981, pp. 264-273

7. Tanenbaum, A. S.: Computer Networks, Prentice-Hall, Englewood Cliffs, N.J. , 1981, Ch. 1

8. Ibid 1, Ch. 2

9. Bhargava, Bharat: "Performance Evaluation of the Optimistic Approach to Distributed Database Systems and it Comparision to Locking," Proc. of 3rd Int. Conference on Distributed Computing Systems, Oct. 18-22, 1982, pp. 508-517

10. Flynn, M. J.: "Some Computer Organizations and Their Effectiveness," IEEE Trans. on Computers, Sept. 1972, pp. 948-960

11. Wittie, L. D.; Van Tilborg, A. M.: "An Introduction to Network Computers," Proc. of ACM82 Conference, Oct. 1982, pp. 199-206

12. Soi, I. M.; Aggarwal, K.K.: "A Review of Computer-Communication Network Classification Schemes," Computer Networks: A Tutorial, Eds. M. Abrams & I.W. Cotton, IEEE Computer Society Press, Silver Springs, Md., 1984

13. Gligor, V.: "Notes on Distributed Operating Systems," Un. of Maryland Used for Course ENEE 748, 1984

14. Myer, T. H.; Vittal, J.J.: "Message Technology in the ARPANET," Proc. of IEEE National Telecommunication Conference '77, Dec. 1977

15. Comer, D.: "The Computer Science Research Network: A History and Status Report," Comm. of ACM, Vol. 26, No. 10, Oct 1983, pp. 747-753

16. Schroeder, M. D.; Birrell, A. D.; Needham, R. M.: "Experience with Grapevine: The Growth of a Distributed System," ACM Trans. on Computer Systems, Vol. 2, No. 1, Feb 1984, pp. 3-23

17. Swann, R. J.; et. al.: "The Implementation of the Cm* Multi-microprocessor," AFIPS Conf. Proc., NCC 1977, pp. 645-655

18. Despain, A. M. Patterson, D. A.: "X-TREE: A Tree Structured Multiprocessor Computer Architecture," Proc. Fifth Ann. Symp. on Computer Architecture, 1978, pp. 144-151

19. Jordan, H.: "A Special Purpose Architecture for Finite Element Analysis," Proc. 1978 Conf. on Parallel Processing, Aug. 1978, pp. 263-266

20. Kapron, F.P.: "Fiber-Optic System Tradeoffs," IEEE Spectrum, Vol. 22, No. 3, March 1985, pp. 68-75

21. Enslow, P. H.: "What is a "Distributed" Data Processing System?," Computer, Vol. 11, Jan. 1978, pp. 13-21

22. Coffman, E. G.; Denning, P. J.: Operating Systems Theory, Prentice-Hall, Englewood Cliffs, N.J., 1973

23. Siework, D. P.: "Process Coodination in Multimicroprocessor Systems," Microarchitecture of Computer Systems, Eds. R.W. Hartenstein & R. Zaks, North-Holland, 1975, pp. 1-8

24. Joseph, M.; Prasad, V. R.; Natarajan, N.: A Multiprocessor Operating System, Prentice-Hall, Englewood Cliffs, N.J., 1984, pp. 50-51

25. Allchin, J. E.: "An Architecture for Reliable Decentralized Systems," Ph. D. Dissertation, Tech. Rpt. GIT-ICS-83/23, Georgia Inst. of Tech., Sept. 1983

26. Weir, D. F.; Holmblad, J. B.; Rothberg, A. C.: "An X.75 Based Network Architecture," Computer Networks: A Tutorial, Eds. M. Abrams & I.W. Cotton, IEEE Computer Society Press, Silver Springs, Md., 1984, pp. 231-241

27. IEEE Standard Digital Interface For Programmable Instrumentation, IEEE Std 488-1978, New York, 1978.

29. Birrell, A. O.; Nelson, B. J.: "Implementing Remote Procedure Calls," ACM Trans. on Computer Systems, Vol. 2, No. 1, Feb. 1984, pp. 39-59

30. Brownbridge, D. R.; Marshall, L. F.; Randell, B: " The Newcastle Connection or Unixes of the World Unite," Software Practice and Experience, Vol. 12, No. 2, Dec. 1982, pp. 1147-1162

31. Haerder, T.; Reuter, A.: "Principles of Transaction-Oriented Database Recovery," Computing Surveys, Vol. 15, no. 4, Dec. 1983, pp. 289-317

32. Batcher, K.P.: "Design of a Massively Parallel Computer," IEEE Trans. on Computers, Vol C-29, No. 9, Sept. 1980, pp. 836-841

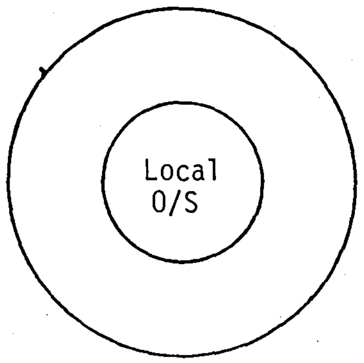
33. Ibid 7, Ch 8.

34. Jensen, E. D.; et. al.: "Reconfiguragble C(2)DDP System," Interim Tech. Report for RADC, Carnegie-Mellon Un., Dec. 1984

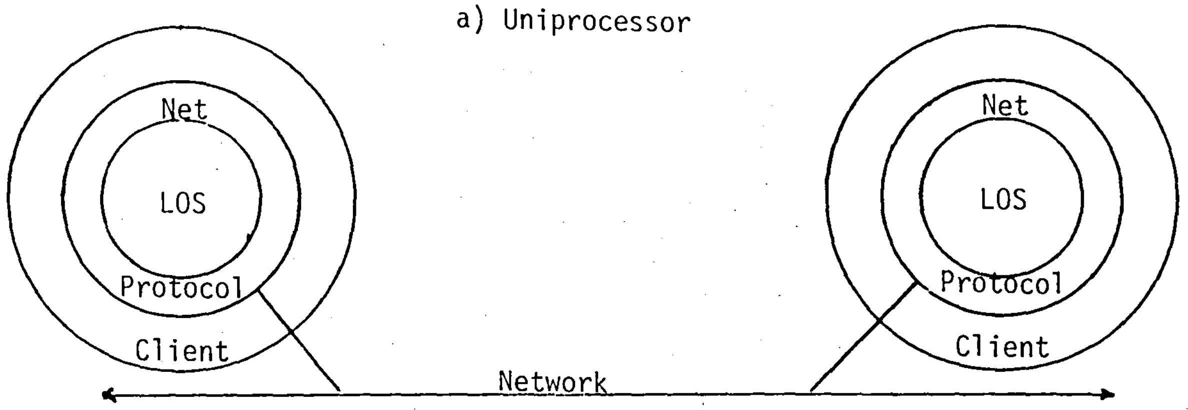
35. Ellis, C. S.; Floyd, R. A.: "The ROE File System," Third Symp. on Reliability in Distributed Software & Database Systems, IEEE Computer Soc. Press, Oct. 17-19, 1983, pp. 175-181

36. Spector, A. Z.; et. al.: "Support for Distributed Transactions in The TABS Prototype," Fourth Symp. on Reliability in Distributed Software & Database Systems, IEEE Computer Soc. Press, Oct. 15-17, 1984, pp. 186-206

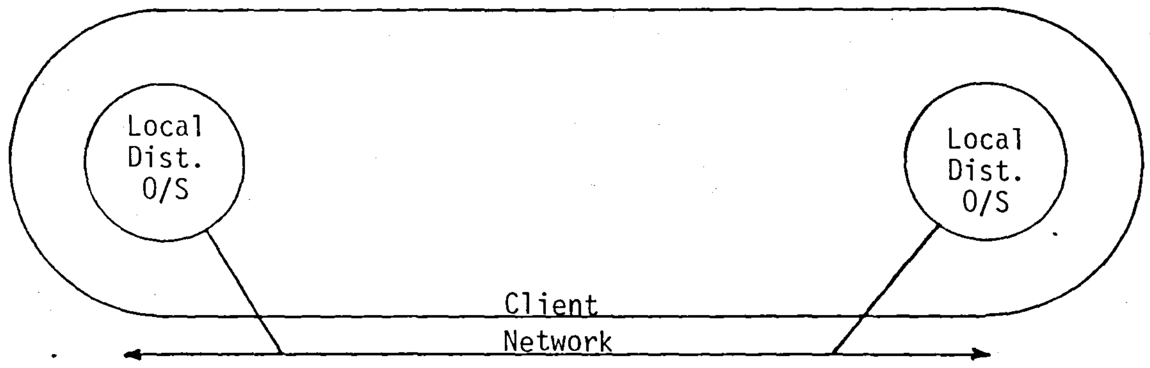
Figure 1.- Computer System Structures



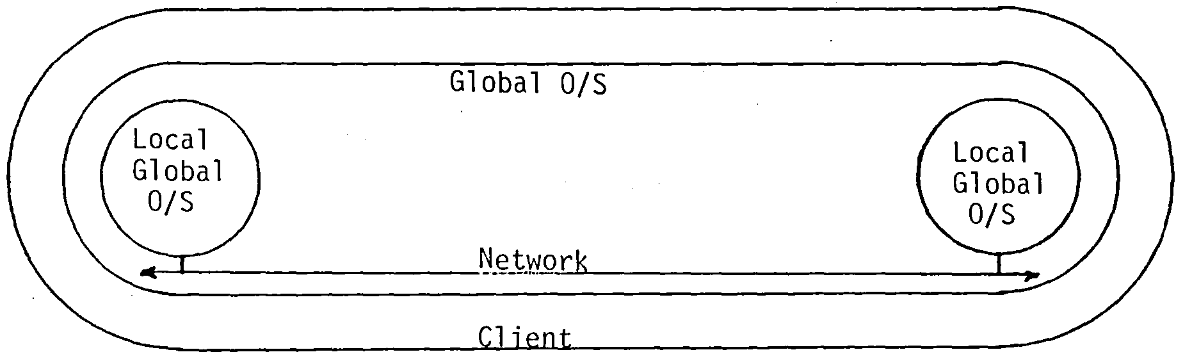
a) Uniprocessor



b) Computer Network



c) Network Computer



d) Multiprocessor - Algorithm - Data Control Computer

1. Report No. NASA TM-86438	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A New Taxonomy for Distributed Computer Systems Based Upon Operating System Structure		5. Report Date June 1985 6. Performing Organization Code 505-37-03-01
7. Author(s) Edwin C. Foudriat		8. Performing Organization Report No.
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665		10. Work Unit No. 11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Technical Memorandum 14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The paper considers characterisitcs of the resource structure found in the operating system as a mechanism for classifying distributed computer systems. Since the operating system resources, themselves, are too diversified to provide a consistent classification, the author examines the structure upon which resources are built and shared. O/S resources must run concurrently and determinately. The underlying structure to accomplish this is the indivisibility or atomic activity provided by the O/S kernel and data structure. The location and control character of this indivisibility provides the taxonomy for separating uniprocessors, computer networks, network computers (fully distributed processing systems or decentralized computers) and algorithm and/or data-control multiprocessors.

The taxonomy is important because it divides machines into a classification that is relevant or important to the client (user of the machine) and not the hardware architect. It also defines the character of the kernel O/S structure needed for future computer systems. For example, based upon the kernel analysis, it is apparent that computer networks are easy to develop from uniprocessors in a value-added fashion, but that true network computers are not yet available because development by extending computer network operating systems is not practical. The paper also discusses in detail what constitutes an operating system for a fully distributed processor.

17. Key Words (Suggested by Author(s)) Distributed computers, Network Computers, Computer Taxonomy, Operation Systems		18. Distribution Statement Unclassified - Unlimited Subject Category 62	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 20	22. Price A02

N-305

End of Document