QUAESTIONES INFORMATICAE

Vol. 1 No. 1

June, 1979

Quaestiones Informaticae

An official publication of the Computer Society of South Africa
'n Amptelike tydskrif van die Rekenaarvereeniging van Suid-Afrika

Editors: Dr. D. S. Henderson,
Vice Chancellor, Rhodes University, Grahamstown, 6140, South Africa.
Prof. M. H. Williams,
Department of Computer Science and Applied Maths,
Rhodes University, Grahamstown, 6140, South Africa.

Editorial Advisory Board

PROFESSOR D. W. BARRON
Department of Mathematics
The University
Southampton S09 5NH
England

PROFESSOR K. GREGGOR
Computer Centre
University of Port Elizabeth
Port Elizabeth 6001
South Africa

PROFESSOR K. MACGREGOR
Department of Computer Science
University of Cape Town
Private Bag
Rondebosch 7700
South Africa

PROFESSOR G. R. JOUBERT
Department of Computer Science
University of Natal
King George V Avenue
Durban 4001
South Africa

MR. P. P. ROETS
NRIMS
CSIR
P.O. Box 395
PRETORIA 0001
South Africa

PROFESSOR B. VON SOLMS
Department of Computer Science
Rand Afrikaans University
Auckland Park
Johannesburg 2001
South Africa

PROFESSOR G. WIECHERS
Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

MR. P. C. PIROW
Graduate School of Business Administration,
University of the Witwatersrand
P.O. Box 31170
Braamfontein 2017
South Africa

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R2	$3	£1.50
Institutions	R4	$6	£3.00

Quaestiones Informaticae is prepared for publication by SYSTEMS PUBLISHERS (PTY) LTD for the Computer Society of South Africa.

Distributed Computer Systems — A Review

N. J. Peberdy

Dept. of Electrical Engineering, University of the Witwatersrand, Johannesburg, South Africa.

Abstract

The past five years have seen a dramatic changeabout in traditional hardware/software relationships: hardware costs have plummeted, and the size, environmental requirements and reliability of computing elements have altered drastically. It now becomes feasible to distribute a computing system, such that processors may be placed adjacent to the processes they control. These distributed computing modules operate in an essentially parallel mode, but are required to communicate in order to co-ordinate their activities. Reliable, secure communication systems must be established to ensure correct operation. Such systems are not only functions of the electrical hardware employed, but also of the software support provided. Of vital importance are the protocols selected, which define and detail an agreed procedure for the exchange of information.

This paper reviews the fundamental software considerations in the design of computer networks, with specific relevance for process-control applications. It discusses in detail, inter-connection strategies and protocols and briefly examines currently adopted schemes. The implications of fully decentralized system control are considered. Of particular concern is the question of the production of reliable, fault-tolerant, secure systems.

1. Introduction

Interest in multiple-processor systems has been generated by the quest for improved system performance. In the past the high cost of processors has limited investigation into such systems. However, due to the recent dramatic fall in hardware costs and processor costs in particular, distributed systems are becoming increasingly feasible. For example, one finds networks of computers being employed in large data-base systems to enable pooling of resources, or permit access to common data-bases. State-of-the-art mainframe computers typically consist of a number of processors, each dedicated to a particular task such as I/O, arithmetic processing, memory management, etc. In the process-control environment, the prospect of improved system performance has led to many so-called distributed control systems. In such a system the plant is partitioned into groups of concurrent tasks, each being controlled by a separate processor, with interconnection to enable co-ordination on a global basis.

The variety of ways in which computers may be interconnected has led to a problem of semantics. Terms such as distributed processing, distributed computers, networks, multiprocessors, etc., are being used in an almost random fashion to describe fundamentally different systems. This is because distributed computers involve new concepts and ideas which have yet to be clearly defined and resolved. For the purposes of this paper, we define a "distributed system" to be a multiplicity of computers that are physically and logically connected together and which co-ordinate their activities on a global basis under centralized or decentralized system control. The term "fully distributed computer" refers to a distributed system in which overall executive control is fully decentralized. The significance of these definitions will become apparent as the paper progresses.

We restrict our interest to those systems in which several processors are physically separated and connected together by data links. Further, we are restricting our area of application to those systems in which the various computer nodes co-ordinate so as to exercise overall control of some system which is itself complex and physically distributed.

2. Requirements of Real-Time Control

In real-time control systems, performance is measured more in terms of response-time than throughput. Events tend to occur in bursts which cause a large and immediate increase in processing activities. This occurs for example, when the plant moves towards a boundary condition. Failure to service a time-critical event within a specified time period could constitute a system failure.

Reliability is one of the most important criteria of plant control. This involves a number of aspects. The control system hardware and software should be correct, that is, error free. However, correctness is not a sufficient condition for reliability because of the finite probability of hardware malfunction. Reliability, therefore, also encompasses the concept of fault-tolerance, whereby the system continues to function in the presence of faults. In practice it is not possible to guarantee correctness particularly in the case of software. "Survivability" of the system involves confinement of initial errors, detection and diagnosis of failures, and finally recovery whereby further damage is prevented and the system is returned to a stable, consistent state.

During design of a plant control system, specifications are often changed, possibly due to changes in the plant itself. In addition, changes are invariably necessary during the life of a plant. The control system must be amenable to such changes. This calls for a highly modular structure both in hardware and software, so that changes in one module can be made without affecting other modules to any great extent.

Modularity involves another aspect. In uniprocessor control systems, each design tends to be different from others. This is especially true if one compares large and small systems. However, in a highly modular distributed system, large and small systems can be configured with the same basic design by varying the number of modules.

The final criterion is cost: any proposed system must be cost effective.

3. Uniprocessor Real-Time Control

Before the advent of low cost processors, plant control was (and in the majority of cases still is) exercised by a single computer. The computer performs all data acquisition, processing, and storage for logging purposes.

It also implements the control function for several processes, which must essentially execute in parallel. In order to perform these different tasks, the computer requires a large, complex executive to allocate CPU time to different tasks. Requests for service are typically asynchronous. If these requests are time critical then either polling at regular intervals is required, or interrupts must be permitted.

Operating systems are notoriously complex structures. The statement by Dijkstra that testing of software can only show the presence of bugs, not their absence, unfortunately sums up the situation. The inherent unreliability of this large software structure is further complicated by interrupts. Such a complex structure can have only very limited reliability.

At a more abstract level, the software designer constructs a virtual architecture which consists of a number of processes, each of which controls one or more plant processes. However, there is no structural correspondence of this virtual multi-purpose- architecture, to the actual uniprocessor architecture [40]. The mapping from the virtual to the real architecture must invariably introduce implicit relationships between processes which were not envisaged by the software engineer. Thus interaction may occur at random, causing very obscure errors.

This paper was presented at the SACAC symposium on Real-time Software for Industrial Applications in Pretoria on 29-30 November, 1978.

All-in-all, the multiprogrammed uniprocessor can have only limited reliability, regardless of speed or computational power. The solution must be found in other areas.

The reason for the complexity of the uniprocessor software is that in the past processors were expensive, and therefore had to be utilized to the full. However, hardware costs have plummeted whereas software costs are rising. It no longer makes sense to maximise processor efficiency. Instead, attempts should be focused on minimising and simplifying software, possibly at the expense of hardware.

4. Distributed Real-Time Control

Due to cost pressures and the requirement for greater reliability, multiple-processor systems have generated considerable attention in the process-control area. At this stage, the state-of-the-art is undeveloped and it will be many years before the potential advantages of distributed systems are fully exploited. However, even now, there are advantages in choosing such a system.

Perhaps the most typical distributed system consists of a number of small processors connected into an hierarchical network under centralized control. The plant is partitioned into a number of concurrent processes. Several processors are then dispersed around the plant, each controlling a process, or a group of processes in close physical proximity. The processors are connected into a network by cables to enable communication. These localized control processors then perform local data acquisition and control. The central computer is freed of the laborious task of data acquisition, and of much of the detailed control function. Its typical function would be to calculate set-points for the other nodes on the basis of selected data transmitted to it from the remote nodes. It would also provide a centralized data logging facility and serve as an access point to the network. Because of the relative simplicity of the tasks it must perform, the central computer could itself be a small computer. A back-up computer could then be economically provided. This would reduce the vulnerability of the system to a fault at the top of the hierarchy.

There are many advantages to this approach. Reliability is greatly improved, due to redundancy of processors, hardware, and communication paths. The failure of a processor for example, is not catastrophic to the system, since the rest of the system may continue to function, possibly with slightly degraded performance. Data is potentially more secure than in a centralized data base system, since it is fragmented over a number of independent memories. Access to each data bank is controlled by its processor. This serves to confine errors. Loss or corruption of a data bank is not catastrophic to the system as a whole. This is true even in the case of the central control computer data, since the data is essentially replicated in the remote nodes.

Apart from reliability, the other main advantage is modularity. Since the system consists of a number of identical processor-memory pairs, it can be constructed to meet the present needs of the user, and system performance may be expanded at a later date in relatively small increments, with correspondingly small and smooth cost increments. Also, changes in one node will have minimal effect on other nodes.

5. Advantages of Distributed Systems

Reliability. As stated earlier, reliability results from redundancy of processors, memory and communication paths. Failures then tend to result in "graceful degradation", whereby the remaining elements may take over the functions of the failed module(s). Performance may be degraded, but the system continues to function. Obviously the tasks performed by many of the processors are not transferable since, for example, a section of the plant will usually (but not necessarily) be interfaced to only one processor. In such a case, the malfunctioning section could be manually controlled until a repair has been made.

Another aspect of reliability is that of communication. Because communication paths have intelligent hardware at each end, highly reliable methods can be used to reduce the probability of errors. Also, because there should always be at least two paths between any two nodes, a break in one path would not then terminate communication.

Response. Response time in a distributed system is potentially faster than a multiprogrammed uniprocessor. This is particularly true if a match of one process per processor is made, thus eliminating multiprogramming and the need for context-switching. A processor could then be dedicated to monitoring a time-critical process, to provide an "instant" response. Whilst this fast response time is at the expense of processor efficiency, software is greatly simplified.

Modularity. A high degree of modularity of hardware and software permits large and small systems to be configured with the same basic design, simply by varying the number of modules. Obviously the degree of modularity is highly dependent on the interconnection topology: this is one of the major considerations in selecting a configuration. A modular system can be upgraded in relatively small performance and cost increments. Also, changes can be made in a module with minimal effect on the rest of the system. This is in contrast to a uniprocessor system, where modification or expansion demand extensive system changes, perhaps even replacement of the computer.

Homomorphism. This term, introduced by Jensen, describes the structural correspondence of the multi-process architecture to the multi-processor architecture. This aspect is responsible for the relative simplicity of the distributed system (at least from this perspective), since software mapping from the virtual to the real architecture is minimal. Reliability is greatly benefitted, since errors are confined to a subset of system functionality and performance [40].

Software. Each processor in a distributed system is controlling one process, or a number of processes of which no more than one is active at any time. Software for such a system will be far simpler than that of a multi-programmed computer. Since system software consists of a number of identifiable modules, it is readily amenable to development by a team. Testing of each module is facilitated. Finally, simplicity in software greatly enhances reliability.

The programmes and data in any one processing-element are relatively isolated from the rest of the system. The probability of illegal interference between processes is thus minimized, if not excluded, because access cannot occur without the knowledge and permission of that processing-element. Data and programmes are thus considerably more secure than in the uniprocessor.

Cost. In a uniprocessor system, the vast network of cables is expensive. A distributed system offers considerable savings in cable costs since all information can be transmitted over a single pair of wires. However, this is complicated by the frequent requirement for manual back-up control from the control room. Also the saving in cable costs is generally a small percentage of total plant cost.

Maintenance. A node can be isolated from the rest of the system in order to perform routine testing and maintenance, without interference to or from the plant.

6. Interconnection Structures

There are obviously many ways in which a number of processors and memory modules can be interconnected. Broadly speaking, multiple-processor systems can be partitioned into tightly- and loosely-coupled systems [18].

Tightly-coupled Systems. A tightly-coupled system is one in which several processors are in close physical proximity. Intercommunication occurs via shared memory or over high-speed parallel buses.

Loosely-coupled Systems. These are systems in which a number of remote computers are connected into a network by data links. There are basically two types of networks [18].

(a) A so-called general purpose network in which each node is capable of independent, stand-alone operation. A typical example is the ARPA network which consists of a number of remote computers. These are linked together into an irregular network to enable sharing of resources [55-58].

(b) A control network of the type discussed earlier. In this system the nodes co-operate in order to achieve overall supervision of a physical system, such as an industrial process.

Interconnection structure is a most important issue in the design of a distributed system, since all other issues tend to be highly dependent on it. In terms of reliability, for example, the question of distribution strategy is critical. In, say, a star-connection, all communication is routed via a single, centralized switch — the whole system is then highly vulnerable to a switch failure.

Due to the current confusion in distributed systems, there is little consensus on the classification of difference topologies. The most widely accepted taxonomy (naming scheme) is that of Anderson and Jensen, shown in Figure 1. The strategy they have adopted in classifying a system is based on the design decisions implicit in the particular configuration [1].

The following section describes the features of various topologies with particular reference to their suitability to real-time plant control.

7. Network Configurations

Fully Connected. Each node is connected to every other node in the network. Cost-modularity is very poor since the addition of one node to an n-node network, requires additional n-connections, one to each node. Complete interconnection is attractive only for very small systems consisting of about three computers. Larger systems are of theoretical interest only.

Partially Connected. For a large number of geographically distributed computers, a partially connected irregular network is most popular. The foremost example is Arpanet which connects over 50 centres spread across North America, with satellite links to London and Hawaii.

Shared Bus. In the shared bus topology, several processing-elements (processor-memory pairs) communicate via a shared bus. Since only one device may transmit over the bus at any one time, bus allocation is of critical importance. Bus control may be centralized by means of some sort of central switch. Decentralized bus control is however preferable from the point of view of reliability, since a failure of the central switch would be catastrophic.

Nodes in a shared bus architecture are an "equal distance" from each other from the point of view of message transfer time. Also, nodes are not distinguished topologically. To increase performance, additional processing-elements may be connected to the bus. However, bus bandwidth is a limiting factor. Unless a fully redundant bus is provided, the system is highly vulnerable to a bus-failure. On the other hand, failure of a processing-element should have minimal effect on system operation, and reconfiguration would simply consist of reallocation of the processes that were being executed by the failed processing-element.

The common bus structure is particularly suited to systems in which executive control is fully decentralized. Honeywell's Modular Computer System is such an example [38, 39]. In the Modular Computer System all inter-process communication is in the form of explicit messages which are transmitted onto the bus, even if the source and destination processes are resident in the same processing-element. This externalization of all inter-process communication not only greatly simplifies software, but also permits all processing-elements to "listen-in" to all conversations. This means that from the control aspect all processing-elements will have the same "view" of the system. This is particularly significant for detection and diagnosis of errors. It also increases the probability that nodes will act in co-operation rather than in conflict. This latter possibility is a real danger in any system in which control is decentralized.

Loops. This topology consists of a number of nodes connected into a loop. Due to the complexity of bi-directional loops over uni-directional loops, traffic in most cases is uni-directional. Messages are placed onto the ring by the source node and circulate around the ring to the destination nodes, being buffered by intermediate nodes. In the Distributed Computing System at the University of California [41-45], a message continues around the loop until it reaches the node which sourced it, where it is removed. All messages thus pass through all nodes. A feature of the Distributed Computing System is that messages are addressed to processes, not to processors. Each "Ring Interface", which is a hardware unit front-ending each computer, has an associative store which holds the names of all the processes currently active in that node. As the message "passes through" the Ring Interface, it checks the message destination name against the process names in its associative store. If there is a match, it copies the message. This technique allows communication to be independent of the number of nodes in the system and allows processes to move freely between processors — this migration being transparent to other processes. The Distributed Computing System is another of the few attempts to decentralise executive control.

Star. Each processing-element is connected by a bi-directional data link to a central switch. The switch accepts a message, performs address translation, and routes the message to its destination. The system is vulnerable to a switch failure; the switch is also a potential bottleneck.

Shared Memory. A system in which two or more processors share access to common memory is termed a "multiprocessor" [4]. Because a single memory constitutes a potential bandwidth problem, it is often fragmented into a number of independent memory modules, to permit several processor-memory accesses to take place simultaneously. There are basically three methods of interconnecting several processors to several memory modules, namely, via a cross-bar switch, or over one or more time shared buses, or by using multi-port memory modules. [3].

The main problem of shared memory structures, is the fact that a processor can access common memory without the knowledge of the other processors. It is possible for a processor to corrupt programmes and data and thereby interfere with the rest of the system. Error confinement is thus a major problem. Complicated protection structures have been devised in an attempt to take care of the problem [30, 35, 36, 49, 52].

8. Communication Concepts

In this section, some basic communication concepts are discussed.

A **message** is a logical unit of information such as a file, a program or an hourly report, which is transferred from one process to another. A message is of variable length, usually with a maximum length which is fairly long (approximately 8k words in Arpanet).

A **packet** is the basic unit of information in the communication subnetwork. Packets may have fixed length, or variable length with a maximum length which is relatively short (approximately 1 000 bits in Arpanet).

A **circuit-switched** network is one in which a complete connection is made between source and ultimate destination before transmission begins. This is the case in a typical telephone network.

A **packet-switched** network is one in which packets are transmitted into the network. The packet contains routing information such as source and destination addresses, as well as error-detection information.

Store-and-Forward. In most packet-switched networks, direct links between all possible senders and receivers do not exist. Packets must therefore pass through intermediate nodes, where they are stored and then forwarded to the next node in the general direction of the destination.

Broadcast Message. A concept that is of considerable value in certain network structures is that of the broadcast message whereby a message is broadcast to all nodes. It is then the responsibility of each node to determine whether the message applies to them. This has the advantage that no routing is required.

Protocol. Protocol is the procedure for the exchange of information between processes. There are four levels of protocol: the process level at which processes communicate; the message level; the packet level; and the hardware level [18]. Here, we are not concerned with the content of messages, only with providing a certain level of confidence that messages will be reliably transported from sender to receiver, with no errors introduced.

Levels of Protocol

(a) **Process Level.** A number of application-orientated processes reside at each node. To enable overall plant control, information such as setpoints, hourly report data, etc., is passed between processes. The information is passed in the form of messages. Messages may also consist of programmes and files which may be transferred to other processors during reconfiguration or to achieve load balancing.

(b) **Message Level.** The message level interfaces between the process level and the communication sub-network. It is the responsibility of the message level to break down long messages into packets. Each packet is then independently routed to the destination node. The message level is then responsible for re-formatting the incoming packets into the original message, which is then passed up to the process level. An error-check field is appended to the message by the sender. If the receiver detects an error in the message, it ignores it, and typically will send off a request to the sender to retransmit the message. If no errors are detected, an "acknowledge message" is sent to the source. This handshake technique gives positive confirmation to the source that the message has reached its destination. The source will retransmit the message if it does not receive a response within a time-out period, with a maximum of about three transmissions. If possible, the retransmissions should take a different route to optimise the chances of the message reaching its destination. The message protocol must be able to cope with error conditions. This involves detection, diagnosis, and recovery. For example, an acknowledge message could get lost, resulting in two identical messages being received by a process. An error in a destination address could send the message to a wrong process. Generally the message header is protected by a separate error-check field, to reduce the probability of undetected errors in the important routing information. If an error is detected in the header then the message is ignored.

(c) **Packet Level.** Due to the difficulty of dealing with long variable length messages (for example, in a forward-and-store network it would be difficult to estimate buffer requirements, and to estimate delays across the network), messages are usually broken up into a number of packets which the network can handle more easily. In a store-and-forward network, each packet is contains its source and destination node. A packet is routed to a node which examines its destination address. If it is not for that node, it is routed to the next node in the general direction of the destination node. This implies a fair degree of intelligence and storage in each node. It also implies that many packets may be circulating at one time. Also, if a message is broken up into a number of packets, these will have to be numbered so that they can be reassembled at their destination into the original message. Each packet might take a different route, so they might arrive out of sequence. As in the case of messages, packets are acknowledged or retransmitted either on request or after a time-out period.

Problems experienced at the message level are also reflected at the packet level. An additional problem can arise — that of deadlock. As there is a finite amount of buffer space at each node, a situation could arise that all buffer space is filled, so that no node can accept another packet. Thus each may wait indefinitely for the others to empty a buffer.

(d) **The Hardware Transmission Level.** This level deals with connection, transmission and signalling methods.

Discussion. Each level interfaces with the level above and below. Provided these interface specifications are adhered to, changes may be made at any level without effect on other levels. Each level must obviously reflect the overall system philosophy.

In certain applications, messages will not be very long and will not need to be broken into shorter packets. The distinction between message and packet levels then falls away. The software/hardware ratio as described here is not necessarily true for all systems. For example, in the Distributed Computing System, the routing function of a node (which consists of copying messages which are destined for local processes, as the message passes through the node), is implemented in hardware. So is the error-checking and acknowledgement of messages. In the Modular Computer System, the hardware is responsible for even more; in fact everything from the message level down. As hardware becomes more powerful and flexible, it is very likely that it will take over much of what has traditionally been done in software.

9. Design Issues

Introduction

It is impossible to consider any single design issue in isolation, since all issues impact on each other. Ultimately, the single criterion is cost — the minimum cost to achieve certain specifications. Because of the present shifting balance of hardware/software costs, it is becoming necessary to reconsider previous decisions. Also, new technology has made possible designs which in the past were not cost-effective. The falling cost of processors in particular has led to fundamental changes in the approach to system architecture in certain application areas. It is no longer necessary to optimise processor usage; a requirement which, in the past, led to large, complicated operating systems.

The most important design decisions are: how to partition the computational load across many processors; how the resulting processes are to communicate; and what interconnection structure is required to support this communication [5].

Load Partitioning

The decomposition of a system into individual, autonomous processes is critical to a successful design. The theoretical approach is of little value at this stage, especially if optimization is attempted [19]. It would appear that the best approach is one based on experience and intuition. Fortunately, most dedicated real-time systems can fairly easily be subdivided into a number of relatively loosely-coupled, well defined sections. Broadly speaking, the objectives of partitioning should be [14].

(a) To segment the system into separate, distinct, autonomous modules;
(b) To minimise interference between modules;
(c) To minimise communication between modules;
(d) To maximise parallelism and concurrency;
(e) To minimise multiprogramming.

Resource Allocation

In terms of hardware, it is obviously preferable that it be composed of identical modules, particularly in the case of processing-elements. Apart from the obvious cost advantages, software would be fully transferable. This considerably facilitates resource allocation, especially if it is to be done dynamically to achieve load balancing or reconfiguration after a fault. Unfortunately, the partitioning procedure is unlikely to yield similarly sized software modules 'naturally'. The larger modules could be

further decomposed, but this procedure will necessarily increase intermodule communication and further increase the problems of synchronizing modules.

Alternatively, the processing-element can be selected so that it could execute the largest software module. This would result in greatly simplified software, at the expense of hardware efficiency. The most cost-effective solution would probably lie somewhere in between. In an hierarchical structure, the requirement for software transportability would be reduced, since higher level software could not normally be executed by lower level hardware, and vice versa.

Process Intercommunication

Processes must communicate in order to achieve overall control of the plant. The first question to ask is: "What do processors in a real-time control plant say to each other?" Hewlett-Packard propose the following answers [7]:

- Deposit and extract data into and out of a distributed or centralized file system;
- Exchange messages directly between executive-level modules in different nodes;
- Share centralized peripherals for reduced overall system cost;
- Share storage of common executable modules for down line loading and execution;
- Dyamically share computing resources as work loads vary.

In a centralized uniprocessor there are many 'protocols', all of them different. Synchronization is achieved by semaphores or flags. Interrupts provide fast response to time-critical service requests. Data is transferred via (common) memory. Device I/0 employs its own protocol. This proliferation of techniques complicates things enormously. In a distributed environment, it could easily be even more complex, due to the relative isolation of processing-elements and the resulting time delays in communication.

In a loosely-coupled system there is a strong argument for adopting a uniform approach to **all** interprocess communication. This is achieved by means of a message-orientated communication system. Input/Output, synchronization, data transfers, etc., all occur by means of explicit messages between the respective processes. Interrupts could be handled directly by hardware where possible or scheduled by hardware together with other messages.

System Control

The resources of any system must be controlled. So far, very little has been said concerning system-wide executive control and yet this is probably the most important issue in distributed systems, especially in the process-control area. Here we discuss this subject only briefly, due to the fact that very little of a practical nature is known regarding fully decentralized control.

There are four elements that can be distributed: hardware (including processors), processing, data and system control [9]. The hierarchical control system, as described above, distributes hardware, processing and data but control of the system is centralized in a single processor. There is no reason, theoretically speaking, why control too, should not be fully decentralized. There are also very good reasons why we should attempt to decentralize control.

In any system in which processors and data are dispersed, considerable problems arise due to the inevitable time lags in transmitting data from one point to another. The system state as seen by the central control processor (in an hierarchical network) will thus always be out of date and therefore inaccurate or even in error. However, in order to make decisions, it must assume that its state information is valid. Thus it may happen that invalid directives are transmitted to processors lower in the hierarchy. These processors which, on the basis of their more up-to-date information, could determine the directive to be faulty, even dangerous, are powerless to refuse it. However, the main weakness of an hierarchical structure is the fact that the whole system is vulnerable to a fault or failure of the central controlling processor or its software. There is thus an upper bound to the reliability of systems in which there is any degree of centralization of control.

The advantages of fully decentralized control are summarized by Jensen [40].

> "We hypothesize that there is a positive correlation between the amount of decentralized system-wide executive control, and the extent to which the system can achieve certain attributes. Foremost amongst these attributes are — 'extensibility',* 'integrity', and 'performance'."

The concepts of fully decentralized computers have certain fundamental differences to partially decentralized systems. These concepts are briefly discussed in the following section.

10. Fully Distributed Computers

Jensen defines a fully distributed computer to be a "multiplicity of processors that are physically and logically interconnected to form a single system, in which overall executive control is exercised through the co-operation of decentralized system elements" [40].

There are essentially five components to this definition [9]:

(a) The system consists of a multiplicity of physical and logical components that can undertake specific tasks on a dynamic basis to achieve load balancing, or to permit reconfiguration of the system after failure of a subset of its elements.

(b) These resources are physically distributed. Each processor has its own operating system and acts as an autonomous entity. The system components interact via a communication network, which employs a two-party co-operative protocol (as opposed to a two-party master-slave protocol) to achieve transfer of information.

(c) The distributed components are integrated into a single, logical, cohesive entity by means of a high-level operating system, which manages all of the system's physical and logical resources. Each processor may have its own operating system which may be unique. Alternatively, as in the Modular Computer System, each processor may have a copy of kernel logic. These copies execute in parallel in a non-hierarchical fashion. Taken as a whole, these kernel copies constitute the high-level operating system.

(d) System structure is transparent to the user. Services are requested from the high-level operating system by name only and the server does not have to be identified. The user may use any of the physical or logical resources of the system, as if these resources were locally available.

(e) The interaction of all physical and logical resources is that of co-operating, autonomous elements. Master-slave relationships are excluded. This is for the obvious reason that a slave is powerless to refuse directives which it could determine to be faulty or undesirable by virtue of its better knowledge of local conditions. It is thus important that the destination resource should be able to refuse a message or reject a request for service based on the knowledge of its own status.

Fully distributed computers are beyond the current state-of-the-art. Nevertheless, there is intense interest in the possibilities of such systems.

Operating Systems

In any multiple-processor configuration, up-to-date status information is never available at one point. This is due to the inevitable time delays that occur when data is transmitted from one point to another. The operating system must therefore be designed to work with inaccurate or even erroneous status information. This is in contrast to a centralized

*'extensibility' is approximately equivalent to 'modularity'

system where the operating system is assumed to have access to complete and accurate information about the overall system status.

The high-level operating system should exercise control over all of the system's resources. This task is greatly simplified if the system is hierarchical. However, the requirement for strictly non-hierarchical control (that is, there are no master-slave relationships) greatly exacerbates the control problems. Even if the multiple autonomous processes are designed to co-operate, the likelihood of conflicting action is much higher than in an hierarchy.

Synchronization of the system as a whole is complicated by the time lags which are inevitable when two autonomous processes attempt to communicate. The conventional methods used in uniprocessors to synchronize two processes such as flags, semaphores, wake-ups, etc., cannot directly be used, although for example, a message carrying the semaphore can be passed between the processes. This consumes a great deal of processing time quite apart from communication delays. A hardware mechanism of some sort might feasibly reduce delay.

Management of resources is a complex problem. For the system to function successfully, efficient control must be exercised over each and every element of the system. Each component of the system cannot be viewed as an isolated entity, but its relationship with the rest of the system must be carefully considered. Dynamic allocation of resources is thus a complex task, particularly if this is being done as a result of failure of some elements of system. This then impacts on the underlying control algorithm for the plant since changes in the overall control strategy would be required.

11. Conclusion

The field of distributed processing is one of considerable activity at the present time. However, we should not expect immediate, dramatic changes. It will be many years before the full potential of distributed systems is exploited.

12. Acknowledgements

The author wishes to acknowledge the many writers who are responsible for the ideas expressed in the text, Douglas Jensen in particular. He also expresses his sincere appreciation to Ken Behr for his assistance in production of the diagram, and to Mike Rodd for editing the article and for many of the ideas expressed therein. The support of the National Institute for Metallurgy and the University of Cape Town is gratefully acknowledged.

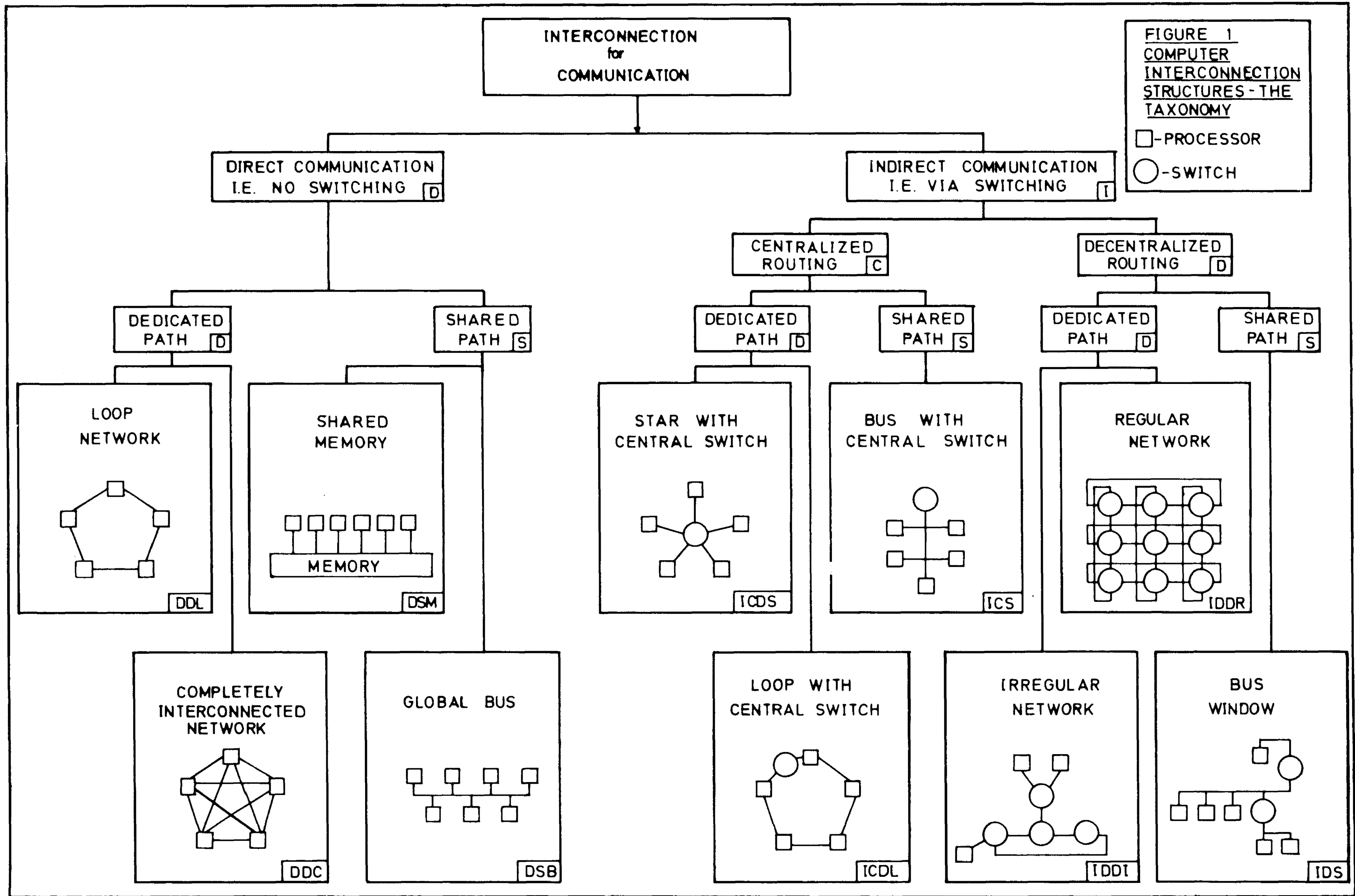

Figure 1

References

Reviews

1. ANDERSON, G.A. and JENSEN, E.D. (1975). Computer Interconnection Structures : Taxonomy, Characteristics, and Examples, *ACM Computing Surveys,* **7,** 197-213.
2. CASAGLIA, G.F. (1976). Distributed Computing Systems : A Biased Review, in *Euromicro,* North-Holland, Amsterdam, 5-18.
3. ENSLOW, P.H. (1977). Multiprocessor Organization — A Survey, *ACM Computing Surveys,* **9,** 103-129.
4. HOOGENDOORN, C.H. (1977). Multiprocessor Systems : Potentials and Problems, paper presented at Program 77 Conference.
5. JENSEN, E.D. (1975). The Influence of Microprocessors on Computer Architecture — Distributed Processing, in *Proc. ACM. Nat. Conf.,* 125-128.
6. RODD, M.G. (1977). A Survey of Distributed Computer Systems with special reference to Industrial Control Applications, Internal Report, University of Cape Town.

Design Issues

7. DICKEY, S. (1976). Distributed Computer Systems — Interfaces, Internals and Inferences in *Infotech State-of-the-Art Report on Distributed Processing,* 225-242.
8. DIJKSTRA, E.W. (1974). Self-Stabilizing Systems in spite of Distributed Control, *Comm. ACM,* **17,** 643-644.
9. ENSLOW, P.H. (1976). What does 'Distributed Processing' Mean?, in *Infotech State-of-the-Art Report on Distributed Processing,* 259-272.
10. LAMPSON, B.W. (1973). A Note on the Confinement Problem, *Comm ACM,* **16,** 613-615.
11. LE LANN, G. (1977). Distributed Systems — Towards a Formal Approach, in *Information Processing 77,* North Holland, Amsterdam, 155-160.
12. METCALF, R.M. (1972). Strategies for Interprocess Communication in a Distributed Computing System, *Proc. Symp. Computer-Communications, Networks and Tele-Traffic,* Polytechnic Institute of Brooklyn.
13. PARNAS, D.L. (1972). On the Criteria to be used in Decomposing Systems into Modules, *Comm. ACM,* **15,** 1053-1058.
14. RAMAMOORTHY, C.V. and KRISHNAROO, T. (1976). The Design Issues in Distributed Computer Systems, in *Infotech State-of-the-Art Report on Distributed Processing,* 377-399.
15. ROBERTS, L.G. (1977). Packet Network Design — The Third Generation in *Information Processing 77,* North-Holland, Amsterdam.
16. SIEWIOREK, D.P. Modularity and Multi-Microprocessor Structures, Dept. of Computer Science and Electrical Engineering, Carnegie-Mellon University, Pittsburgh.
17. SIEWIOREK, D.P. and BARBACCI, M.R. (1976). Modularity and Multi-processor Structures — Some Open Problems in the Construction and Utilization of Mini- and Micro-Processor Networks, in *Infotech State-of-the-Art Report on Distributed Systems,* 403-418.
18. SLOMAN, M.S. and PENNEY, B.K. (1977). Communication Requirements for Microcomputer Networks in Process Control Applications, internal report, Dept. of Computing and Control, Imperial College, London.
19. WHITE, G.N.T. and SIMMONS, M.D. (1977). Analysis of Complex Systems, University of Cambridge, England.

Software

20. ABRAMSON, N. and KUO, F.F. (1973). *Computer Communication Networks,* Prentice-Hall, Englewood Cliffs.
21. ANDERSON, G.A. Interconnecting a Distributed Processor System for Avionics, in *Proc. 1st Annual Symp. on Computer Architecture,* 11-16.
22. BRINCH-HANSEN, P. (1970). The Nucleus of a Multiprogramming System, *Comm ACM,* **13,** 238-250.
23. DAGLESS, E.L. A Multimicroprocessor: CYBA-M, Dept. of Electrical and Electronic Engineering, Univ. College of Swansea.
24. DAVIES, D.W. and BARBER, D.L.A. (1973). Communication Networks for Computers, John Wiley and Sons.
25. DIJKSTRA, E.W. (1968). The Structure of 'THE' Multiprogramming System, *Comm ACM,* **11,** 341-346.
26. DIJKSTRA, E.W. (1970). Notes on Structured Programming, Report 70 — WSK — 03, Technical University, Eindhoven.
27. FARBER, D.J. (1974). Software Considerations in Distributed Architectures, *Computer,* March 1974, 31-35.
28. HARMALA, A.D. (1975). Benefits of Localized Control with microcomputers, *Computer Design,* May 1975.
29. HEWLETT-PACKARD JOURNAL (1978). Articles on the HP Distributed System Network, March 1978.
30. LAMPSON, B.W. (1969). Dynamic Protection Structures, in *Proc. AFIPS FJCC,* 27-38.
31. McFADYEN, J.H. (1976). Systems Network Architecture : an Overview, *IBM Systems Journal,* **15,** 1, 4 - 22.
32. METCALF, R.M. and BOGGS, D.R. (1976). Ethernet: Distributed Packet Switching for Local Computer Networks, *Comm ACM,* **19,** 395-403.
33. MOORE, M.J. (1974). A Distributed Microprocessor System for Avionics, 8th Asilomar Conference on Circuits, Systems and Computers, Dec. 3-5 1974, *Calif. IEEE,* 92-96.
34. MORGAN, D.E. and TAYLOR, D.J. (1977). A Survey of Methods of Achieving Reliable Software, *Computer,* Feb. 1977, 44-53.
35. POPEK, G.J. (1974). Protection Structures, *Computer,* June 1974, 22-23.
36. WULF, W.A. (1975). Reliable Hardware/Software Architecture, *IEEE Trans. on Software Engineering,* **SE-1,** No. 2.
37. YAN, G. and L'ARCHEVEQUE, J.V.R. (1976). On Distributed Control and Instrumentation Systems for Future Nuclear Power Plants, *IEEE Trans. Nuclear Science,* **NS-23,** 431-435.

The Modular Computer System

38. JENSEN, E.D., A Distributed Function-Computer for Real-time Control, in *Proc. 2nd Annual Conf. on Computer Architecture,* ACM, 176-182.
39. JENSEN, E.D. (1976). Distributed Processing in a Real-time Environment, in *Infotech State-of-the Art Report on Distributed Processing,* 305-318.
40. JENSEN, E.D. (1978). The Honeywell Experimental Distributed Processor — An Overview, *Computer* Jan. 1978, 23-38.

The Distributed Computing System

41. FARBER, D.J. (1975). A Ring Network, Datamation, Feb. 1975, 44-46.
42. FARBER, D.J. and HEINRICH, F.R. (1972). The Structure of a Distributed Computing System — The Distributed File System, in *1st International Conf. of Computer Communications,* ACM/ IEEE, Oct. 24-26.
43. FARBER, D.J. and LARSON, K.C. (1972). The System Architecture of the Distributed Computer System — The Communications System, in *Proc. Symp on Computer-Communications Networks and Teletraffic,* Polytechnic Institute of Brooklyn, 21-27.
44. FARBER, D.J. and LARSON, K.C. (1972). The Structure of a Distributed Computing System — Software, in *Proc. Symp. on Computer-Communications Networks and Teletraffic,* Polytechnic Institute of Brooklyn, 539-545.
45. ROWE, L.A. *et. al.* Software methods for Achieving Fail-Soft Behaviour in the Distributed Computing System, Dept. of Information and Computer Science, Univ. of California, Irvine.

C.mmp

46. BELL, C.G. and FREEMAN, P. (1972). C.ai — A Computer Architecture for AI Research, in *Proc. AFIPS FJCC,* 779-790.
47. FULLER, S.H. (1976). Price Performance Comparison of C.mmp and the PDP-10, in *3rd Annual Symp. on Computer Architecture,* Jan. 19-21, 1976, ACM.
48. WULF, W.A. and BELL, C.G. (1972). C.mmp — A Multi-Mini-Processor, in *Proc. AFIPS FJCC,* 765-777.
49. WULF, W., COHEN, E., CORWIN, W., JONES, A., LEVIN, R., PIERSON, C. and POLLACK, F. (1974), HYDRA — The Kernel of a Multiprocessor Operating System, *Comm ACM,* 17 337-345.
50. WULF, W. and LEVIN, R. (1975). A Local Network, *Datamation,* 47-50.

CM*

51. FULLER, S.H. *et al.* Computer Modules — An Architecture for Large Digital Modules, in *Proc. 1st Annual Symp. on Computer Architecture,* ACM, 231-237.
52. JONES, A.K. *et al.* (1977). Software Management of CM* — A Distributed Multiprocessor, in *Proc. AFIPS NCC,* 657-663.
53. SWAN, R.J. *et al.* (1977). CM* — A Modular, Multimicroprocessor, in *Proc. AFIPS NCC, 637-644.*
54. SWAN, R. J. *et al.* (1977). The IMplementation of the CM* Multimicroprocessor, in *Proc. AFIPS NCC,* 645-655.

Arpanet

55. CARR, C.S. *et al* (1970). Host-Host Communication Protocol in the ARPA network, in *Proc. AFIPS SJCC,* 589-597.
56. HEART, F.E. *et al* (1970). The Interface Message Processor for the ARPA Computer Network, in *Proc. AFIPS SJCC,* 551-567
57. ORNSTEIN, S.M. *et al* (1975). Pluribus — A Reliable Multiprocessor, in *Proc NCC,* 551-559.
58. THOMAS, R.H. (1973). A Resource Sharing Executive for the Arpanet, in *Proc. NCC,* 155-163.

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles or articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be in double-spaced typing on one side only of Henderson or Prof. M. H. Williams at

Rhodes University
Grahamstown 6140
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM,* **9**, 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of Context-free Languages,* McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Quaestiones Informaticae

Contents/Inhoud

A hardware-based real-time operating system 1
M.G. Rodd

Real-time interactive multiprogramming 5
A.D. Heher

Distributed Computer Systems — a review 17
N.J. Peberdy

The P-NP question and recent independence results 26
N.C.K. Phillips

Text compression techniques 30
J.E. Radue

Spectrum Offset