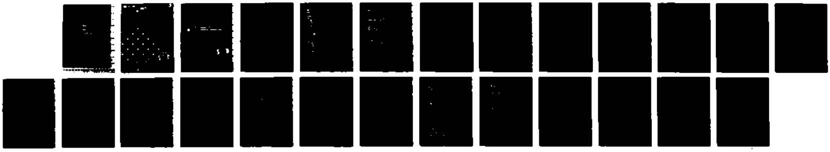
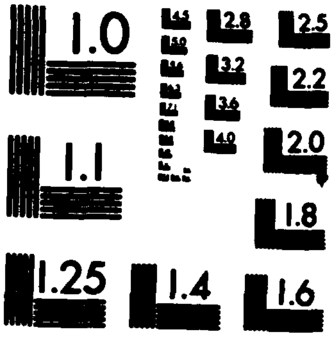
AD-A181 853 DISTRIBUTED SYSTEMS TECHNOLOGY SURVEY(U) 1/1
CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING
INST E C COOPER MAR 87 CMU/SEI-87-TR-5 ESD-TR-87-106

UNCLASSIFIED F/G 12/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

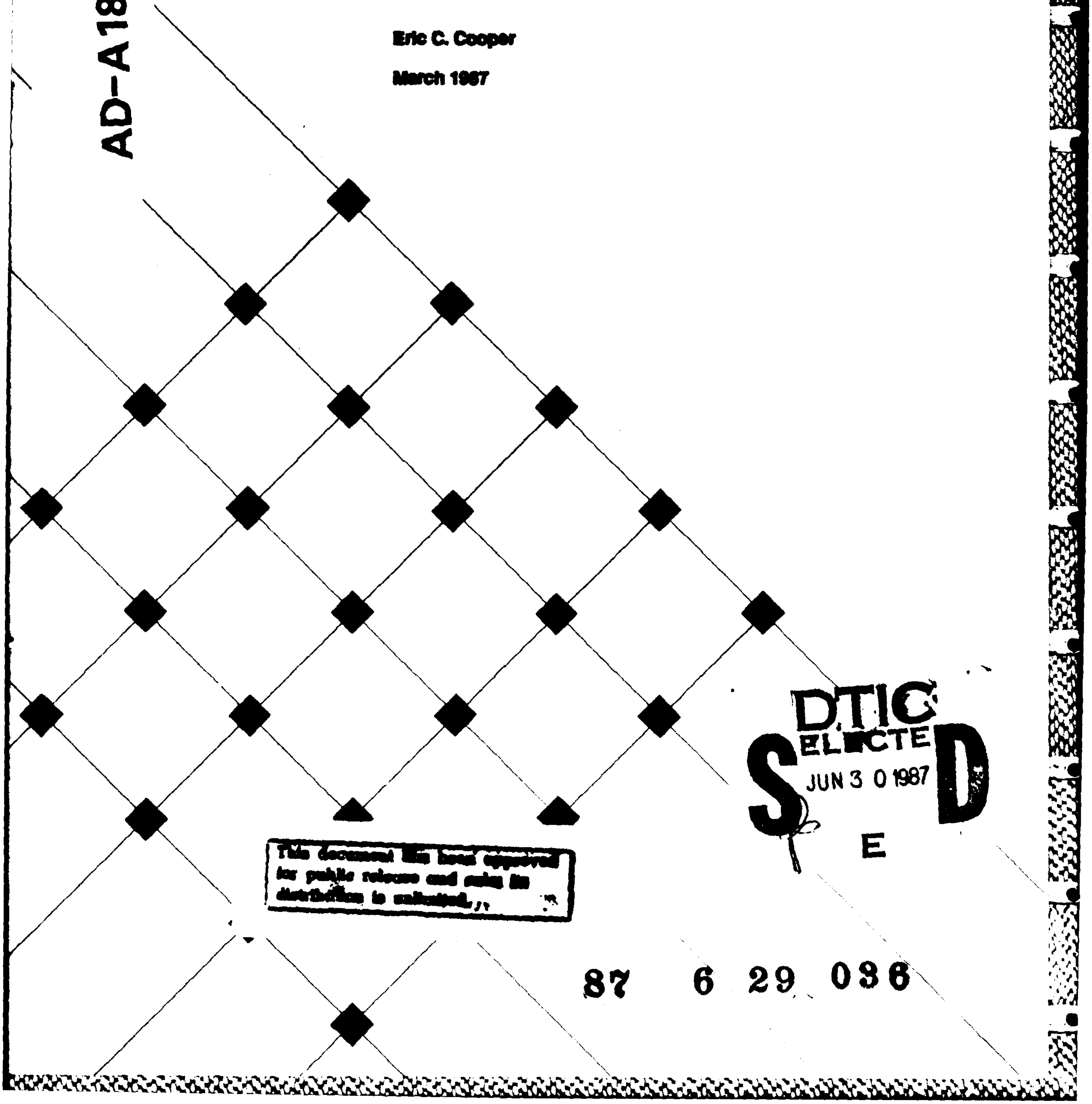
Technical Report
CMU/SEI-87-TR-5
ESD-TR-87-106

Carnegie-Mellon University
Software Engineering Institute

Distributed Systems Technology Survey

Eric C. Cooper

March 1987



2

Technical Report

CMU/SEI-87-TR-5

ESD-TR-87-106

March 1987

Distributed Systems Technology Survey

Eric C. Cooper

Approved for public release.
Distribution unlimited.

Software Engineering Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl H. Shingler
SEI Joint Program Office

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Services. For information on ordering, please contact NTIS directly: National Technical Information Services, U.S. Department of Commerce, Springfield, VA 22161.

A

Table of Contents

Foreword

The Technology Identification and Assessment Project combined a number of related investigations to identify:

- existing technology in a specific problem area to review research and development results and commercially available products;
- new technologies through regular reviews of research and development results, periodic surveys of specific areas, and identification of particularly good examples of the application of specific technologies; and
- requirements for new technology through continuing studies of software development needs within the DoD, and case studies of both successful and unsuccessful projects.

Technology assessment involves understanding the software development process, determining the potential of new technology for solving significant problems, evaluating new software tools and methods, matching existing technologies to needs, and determining the potential payoff of new technologies. Assessment activities of the project focused on core technology areas for software engineering environments.

This report is one of a series of survey reports. It is not intended to provide an exhaustive discussion of topics pertinent to the area of distributed systems technology. Rather, it is intended as an informative review of the technology surveyed. These surveys were conducted in late 1985 and early 1986.

Members of the project recognized that more general technology surveys have been conducted by other investigators. The project did not attempt to duplicate those surveys, but focused on points not addressed in those surveys. The goal in conducting the surveys was not to describe the technology in general, but to emphasize issues that either have a strong impact on or are unique to software engineering environments. The objective in presenting these reports is to provide an overview of the technologies that are core to developing software engineering environments.

1. Introduction

One of the core technology areas in which project members were interested is distributed systems technology. This report surveys the technical issues involved in designing distributed systems, with particular emphasis on those aspects that affect software engineering environments.

Economics is the driving force behind the proliferation of distributed systems. Workstations are a cost-effective way of providing computing power to individuals. Local area networks are a cost-effective way of sharing access to more expensive and less frequently used resources like laser printers and large disks, as well as a means for users to share information.

In distributed systems, however, there is a fundamental dichotomy between the need for integration (to achieve sharing) and the need for autonomy (to control one's local environment). Many of the technical solutions presented here can be evaluated in terms of how they balance these two needs.

Distributed systems have a number of well known potential benefits:

- Modularity: Resources can be added and reconfigured easily.
- Performance: Parallelism can be used to perform jobs more quickly.
- Availability: Redundancy can be used to provide non-stop service.

To achieve these benefits in practice, however, requires solutions for a number of difficult technical problems. Also, there are trade-off relationships between, for example, using a distributed system for increased performance versus using it for increased availability. The following sections discuss some of the important technologies and issues involved in distributed systems. References to appropriate surveys are included in the discussion, but two general references are appropriate here. First, the book edited by Lampson *et al.* [18] is an excellent overview of many aspects of distributed systems. Second, Tanenbaum's book on computer networks [36] is probably the best starting point for more information on networks and protocols.

2. Hardware Technology

Economic factors are a major reason for the proliferation of distributed systems. Processors, memory, and magnetic and optical disks are sufficiently inexpensive to allow an organization to deploy a workstation in every office, in addition to supporting a machine room with mainframes and file servers. Current workstations typically have from 1 to 10 MIPS (million instructions per second) of processing power, 1 to 10 megabytes of memory, 20 to 200 megabytes of disk storage, and cost less than 20,000 dollars.

A variety of network technologies are available for interconnecting these components. A distinction is commonly made between local area and long-haul networks. For both physical and administrative reasons, local area networks (LANs) are typically used within, and managed by, a single organization. Currently, most LANs are constructed from coaxial cable or fiber optics, with bandwidths ranging from 10 to 100 million bits per second and lengths on the order of 1 kilometer. The interconnection topologies of such networks include bus, ring, tree, and star structures.

Long-haul networks, on the other hand, can span continents, and are usually managed by companies or government agencies for use by others. The technologies used for long-haul networks include telephone lines and satellite links. Long-haul networks use one or more of the following switching techniques:

- Circuit switching
- Message switching
- Packet switching

Circuit switching is used in the telephone system. In this scheme, a route through the network is established before any data is sent. Since communication between computers tends to come in bursts, circuit switching does not provide good utilization of available bandwidths, and the time required to pre-allocate a circuit may be unacceptably long.

Message-switched and packet-switched networks are also called store-and-forward networks because data is independently routed from one switching node to another. For reasons of reliability, the network topology of a store-and-forward network should be connected redundantly, so that there are several paths or routes between any two nodes.

In message switching, individual messages are routed from one switching node to another. This eliminates the long set-up time associated with circuit switching and provides better utilization of bandwidths. The disadvantage of this approach is that the variable size of messages makes it difficult to allocate node buffering resources efficiently.

In packet switching, messages are first broken up into fixed-size packets, which are then individually routed through the network and reassembled at the destination. Different packets of the same message may be routed along different paths, and hence may arrive out of order. Higher level protocols are used to handle out-of-order packets. Since all packets are of the same (relatively small) size, buffering at intermediate nodes is simplified. The ARPANET is an example of a long-haul packet-switched network.

3. Internetworks

Because of their low cost, workstations tend to proliferate in organizations, and the need for LANs tends to grow as well. Large organizations are soon faced with the necessity of connecting several LANs through a structure called an internetwork. In fact, some of the components of an internetwork can be long-haul networks. In the DARPA Internet, for example, the long-haul ARPANET is connected with hundreds of LANs at universities and research laboratories. The same store-and-forward approach can be used in an internetwork, viewing the internet gateways as the packet switches of a single larger network.

4. Protocols

Protocols are used to provide virtual communication services with properties different from (and typically at a higher level than) those provided by the physical network. This leads naturally to a layered model of protocols, such as the one that has been standardized in the ISO Reference Model for Open Systems Interconnection [17].

Further discussion about protocols may be found in the survey article by Tanenbaum [37].

4.1. ISO Reference Model

The ISO Reference Model consists of the following seven layers:

1. Physical layer: low-level communication of bits
2. Data link layer: framing, checksumming

3. Network layer: internetwork addressing and routing

4. Transport layer: reliable communication, host-to-host addressing

5. Session layer: connection management, process-to-process addressing

6. Presentation layer: data formatting, encryption, compression

7. Application layer: user programs

The ISO Reference Model does not match all protocol architectures perfectly. In the DoD Internet family of protocols, for example, the IP [26] and TCP [27] protocols provide functionality that ranges from the data link layer to the session layer.

4.2. Transport Protocols

Communication services can be characterized by a number of attributes:

- The need for a connection establishment protocol before communication can occur
- The number of communicating entities
- Reliability of data delivery
- Client interface (messages or stream abstraction)
- Fixed or variable length of messages

The following is a brief characterization of a number of transport protocols according to the above attributes.

- Datagram protocol: connectionless, unreliable delivery, fixed-size packets. Examples include the Xerox PARC PUP protocol, the DoD Internet Protocol (IP), and the DoD User Datagram Protocol (UDP).
- Byte stream protocol: connection-based, reliable delivery, stream abstraction. Examples include the Xerox PARC byte stream protocol (BSP) and the DoD transmission control protocol (TCP).
- Message protocol: connectionless, reliable delivery, variable-length messages. Examples include the protocol used by the Spice system [29].
- Request/response protocol: connectionless, reliable delivery, variable-length alternating request/response messages. Examples are described by Birrell and Nelson [6].

A recent topic of research has been the incorporation of many-to-many communication semantics into various transport protocols [1, 8, 9]. New protocols in each of the above classes will likely be extended with many-to-many semantics.

4.3. Higher-Level Protocols

Higher-level protocols, those implemented at the session layer or higher in the ISO model, are correspondingly harder to characterize. Examples from the DoD Internet family include the Telnet network terminal protocol, the file transfer protocol (FTP), and the mail delivery protocol (SMTP). Other areas of research include protocols for graphics, window managers, voice, multimedia messages, bootstrap loading, remote debugging and monitoring, and remote procedure call, discussed more fully below.

5. Heterogeneity

Currently, there is no single standard machine architecture, operating system, programming language, or programming environment, and such standards are not likely to appear in the near future. As a result, organizations find themselves faced with the problem of integrating a heterogeneous collection of such resources. As evidenced by a recent workshop in Eastsound, Washington, that was devoted solely to the problems of heterogeneity, and by current research projects in heterogeneity at institutions such as CMU and MIT, this is the key problem in distributed systems today [24].

The most important pitfall to avoid in a heterogeneous system is the lowest common denominator effect. This occurs when interfaces are only defined for those operations that are supported by all components in the system. As the number of heterogeneous components increases, this set of common operations may approach the empty set.

A number of techniques can be used to avoid the lowest common denominator effect. One technique is a common data representation protocol, in which all communicating components translate their interactions into a standard external representation. As described below, this can be handled automatically in remote procedure call systems through the use of a stub generator. The main difficulty with this technique is that the representation protocol itself suffers from the lowest common denominator effect. The advantage, however, is that such protocols are flexible since they are capable of representing arbitrary programming language data types like arrays and records. DeSchon surveys a number of data representation standards [10].

Another technique is called option negotiation [34], in which each pair of communicating parties negotiates which protocol options they will support. This approach allows each pair to communicate with maximal functionality. The option negotiation approach is applicable at many levels in a heterogeneous distributed system.

The data representation protocol and option negotiation techniques can be successfully combined. For example, the remote procedure call system at the DEC Systems Research Center uses negotiation at binding time to decide between two possible data representation protocols.

A third and somewhat ad hoc approach to coping with heterogeneity is the proxy technique. A proxy is a specialized agent in a remote environment whose purpose is to provide an interface to that environment that is more compatible with other components of the system. This approach was first used in remote job entry (RJE) systems to access batch facilities from timesharing systems. It has been used successfully in the Locus system [25] to integrate IBM mainframes transparently into a distributed UNIX[1] environment.

[1] UNIX is a registered trademark of Bell Laboratories.

6. Models of Distributed Programs

Although transparency is desirable at the highest levels of a distributed system, at some lower level the fact that the system is distributed must be made available to the programmer. How this is done is largely determined by the model of distributed programs that the systems designer adopts.

One of the most well known approaches, developed at Xerox PARC in the 1970s, is called the client/server model. The computing environment is assumed to consist of personal workstations and a collection of shared network services implemented by server machines. Such services might include file storage (discussed more fully below), printing, and electronic mail. The programs running on the user's workstation are viewed as clients of these servers. The client/server model is a simple extension of the application program/operating system model familiar in centralized timesharing systems. It is flexible because new services are easily added, and it supports a heterogeneous environment well: "Black boxes" can be used as servers as long as some interface can be constructed on the client side. A disadvantage of the client/server model is that it does not support load balancing or multi-machine parallel applications, although such program structures can be shoe-horned into this model by using a pool of "compute servers."

Some of these deficiencies are remedied in the network operating system (NOS) model. In this model, a transparent interface to all network resources is presented to the applications programmer, not just at the user interface level. The Locus system at UCLA [25] and the Spice system at CMU [29] are successful examples of systems that follow this model. A major disadvantage of the network operating system model is its difficulty in accommodating heterogeneity (in the form of black boxes) because it assumes that a common software interface can be installed on all the network resources.

7. Operating System Issues

This section briefly describes a number of operating system features that are particularly important for supporting distributed systems.

A message-based operating system consists of an efficient kernel implementation of processes, virtual memory, and inter-process communication, together with a set of server processes providing conventional operating system services such as device drivers and file systems. The Accent kernel is a prime example of a message-based system [29].

Message-based kernels allow inter-process communication to be extended over the network in a simple and transparent fashion. The key is the notion of intermediary processes that intercept remotely destined messages and perform the appropriate forwarding.

There is growing agreement that a lightweight process mechanism is essential to support commonly used distributed program structures. A number of lightweight processes can share a single address space; this allows the construction of servers, for example, that correctly handle concurrent incoming requests. The lack of such lightweight processes has been a weak point of UNIX and a number of message-based operating systems.

A process migration facility allows a running process to be moved from one machine to another. Such a facility is a valuable mechanism for implementing load balancing policies, whereby jobs are moved off heavily loaded machines and onto lightly loaded ones. Variants of process migration can be used to increase fault tolerance by checkpointing process state. Process migration is greatly simplified in message-based operating systems [28].

A simpler form of load balancing can be accomplished at task creation time by starting the task on a lightly loaded processor. Further experience is needed to determine whether the full power of process migration is necessary.

Workstation technology has advanced to the point where most new high-end workstations are multiprocessors with approximately 10 processors. Operating system support for multiprocessors, and in particular for efficient execution of parallel programs, will be an increasingly important requirement.

Finally, UNIX compatibility is often a practical necessity. The wide variety of software tools available under UNIX would be prohibitively expensive to port to an incompatible environment.

Many of the features mentioned in this section have been included in the design and implementation of the MACH-1 operating system at CMU [3], a kernel and programming environment that will probably serve as the new foundation for DARPA-sponsored research in strategic computing.

8. Programming Language Issues

One approach to integrating distributed programming primitives into the programming environment is to incorporate them into the programming language itself. This approach can be accomplished in two ways: the mechanisms can be built into the language, or they can be provided externally.

CSP [16] and Ada [12] are examples of languages with built-in communication primitives. This approach extends the benefits of strong typing to distributed programs because the language is the only interface to the communication mechanism. Unfortunately, most languages of this type ignore the problem of heterogeneous environments. As discussed previously, in order to cope with heterogeneity, some common data representation protocol or negotiation scheme must be used among the language implementations on different machines. Without a language-defined standard, programs produced by different compilers are unlikely to be able to communicate. Ada provides only a partial solution to this problem in the form of pragma statements that allow control over the representation of data types.

In message-based operating systems, primitives for message communication are typically integrated into the programming language in the form of a subroutine library. Again, little support for heterogeneity has been provided. Issues of data representation and type safety are usually the responsibility of the programmer.

Remote procedure call (RPC) systems represent a compromise between the built-in and the external approach. By using a stub generator, the remote procedure call mechanism can be closely coupled to, yet separate from, the compiler. This approach is described in more detail in the next section.

9. Remote Procedure Call

Remote procedure call is a combined protocol-level and language-level mechanism for constructing distributed programs. A remote procedure call mechanism allows a programmer to write a distributed program in the same way one writes a single-machine program: using procedure calls in one's favorite programming language. Remote procedure call meshes well with both the client/server and network operating system models.

The language-level integration of remote procedure call into a conventional programming language is typically accomplished by the use of a stub generator, a specialized compiler that translates a module interface into stub procedures for the client and server halves of a remote interface. The stub procedures handle the details of representing the data types of the programming language in an external form when they are sent in messages, and the conversion to and from the internal form. The stub procedures also interface with the lower level request/response protocol used to exchange the call and return messages.

The stub generator approach has a number of advantages:

- The stub generator manipulates source-level programs, so strong typing can be provided.
- The stub generator is separate from the compiler, so the same stub generator can be used with any compiler for that language.
- The stub generator is a natural place to "hide" knowledge about the external representation protocols and/or negotiation schemes used between heterogeneous machines.

To invoke a remote procedure, the client stub builds a call message containing the name of the procedure to be invoked and the external representation of its arguments. The client sends the call message to the server machine, where it is interpreted by the server stub. The arguments are converted to their internal representation and are passed to the named procedure. When the procedure returns, its results are externalized in a return message and sent back to the client. Finally, the client stub converts the results back into internal form and returns them to the client program.

Nelson gives a comprehensive treatment of remote procedure call in his thesis [23]. Birrell and Nelson describe the transport protocol and binding mechanisms used in an implementation of RPC at Xerox PARC [6].

9.1. Advantages of Remote Procedure Call

The single biggest advantage of remote procedure call is that it makes writing distributed programs almost as easy as writing single-machine programs. The same software development methodologies that work well for centralized systems, such as the use of modularity, abstract data types, and stepwise refinement, continue to work just as well when extended with remote procedure call.

9.2. Disadvantages of Remote Procedure Call

Although remote procedure call has become extremely popular, it is not a panacea. In particular, it is not suitable for the transfer of large amounts of data, or for communication over high-latency media. Special bulk data transfer protocols are preferred in such cases.

One common criticism of remote procedure call, namely that the synchronous nature of remote procedure call does not allow any parallelism, is really not a problem. In fact, remote procedure call neither helps nor hinders parallelism. The above criticism is usually accompanied by an argument in favor of non-blocking remote calls, where the application can either poll for the return value or have it delivered asynchronously. Such features are actually a poor man's substitute for lightweight processes, and are only desirable in environments where processes are heavyweight and expensive. If lightweight processes are well supported in the programming language and environment, they become the natural means of achieving parallelism in conjunction with remote procedure call. If not, polling or asynchronous delivery mechanisms can be simulated with remote procedure call, but use of such features can result in rather convoluted programs. For the most effective match, systems should support both remote procedure call and lightweight processes.

10. Software Tools for Distributed Environments

Making software tools function transparently in a distributed environment often requires substantial effort.. Consider some of the tools that have become standard equipment in centralized environments:

- Compilers
- Linkers
- Debuggers
- Profiling tools
- Version control and system configuration tools

A number of issues must be addressed when extending these tools to distributed environments.

Programming language compilers and interpreters must be integrated with communication facilities such as message primitives or remote procedure call. The software engineering issues are complicated by machine dependencies, language dependencies, and compiler dependencies, any one of which can effect the representation of programming language data types in messages.

Debuggers must be extended to allow single-stepping across machine boundaries when following a chain of remote procedure calls. It should be possible to set breakpoints in remote modules and to trace the flow of control of a distributed program. An advantage of message-based operating systems for distributed debugging is the ability to encapsulate the entire environment of a process, since all of its interactions occur via messages.

Profiling tools provide the programmer with histograms of where time is spent in a program. This allows the programmer to detect bottlenecks and to apply optimizations where they will do the most good. In the distributed case, profiling must work correctly when portions of the program execute at remote nodes.

Version control and system configuration is a particularly difficult problem in a distributed environment. Schmidt describes a variety of techniques for maintaining consistent releases of large software systems in the Xerox PARC environment [31]. Shared file servers, discussed below, are essential to the success of such a scheme.

11. Security

A distributed environment raises a number of security issues. First, the broadcast nature of most local area networks makes them particularly vulnerable to eavesdropping. Anyone with a personal workstation on an Ethernet can easily monitor all network traffic. Secondly, the lack of control over the software run in an individual workstation makes masquerades, replays, and similar active threats possible.

These problems are solved in single-machine or centralized environments by physical security: locked machine rooms and protected terminal lines. Unfortunately, the decentralized nature of distributed systems precludes such measures. Logical rather than physical schemes must be used instead.

The simplest problem to solve is that of eavesdropping. The solution uses encryption: two persons wishing to communicate do so by encrypting all their messages with a secret key known only to them. This effectively constructs a secure private communication channel on top of the underlying insecure public channel. The Data Encryption Standard (DES) can be used for secret-key encryption and decryption [21]. Hardware implementations of DES are available and should be included in new workstations.

More elaborate encryption-based schemes can be used to solve the authentication problem, in order to prevent masquerades and similar active threats [11, 22]. In such a scheme, a person can securely identify himself to another person by obtaining from a mutually trusted authentication service an "proof of identity" that is unable to be forged. Birrell has described a comprehensive scheme that provides both privacy and authentication for remote procedure calls [7].

The encryption-based schemes that have been proposed in the literature do not afford much protection against denial-of-service attacks. It has been observed that passive threats are difficult to detect but easy to prevent, while active threats are easy to detect but difficult to prevent.

12. Distributed File Systems

Distributed file systems have more impact on programming environments than any other aspect of distributed systems. A good discussion of file servers and distributed file systems may be found in the survey article by Svobodova [35].

12.1. Files and Directories

Files are the primary means of storing and sharing long-lived information in computer systems. File systems may impose structure on the contents of files (index or record structures or file types) or may treat the contents merely as sequences of bytes. This report takes the latter approach and views a file as a sequence of uninterpreted bytes; any structure imposed on file contents is viewed as a logically higher level. A common approach is to deal only in machine-sensible *unique identifiers* at the file system level.

A separate concept, often lumped together with the file system, is the *directory system*, which provides a mapping from user-sensible names to file identifiers. Directories may themselves be implemented as files containing name/identifier pairs. The directory system implements creation, deletion, lookup, and enumeration of name/identifier pairs. Additional functions may include expansion of patterns containing wildcard characters.

The directory system is responsible for any structuring of file names. A common approach is a tree structured directory system, in which the full name of a file is a path name consisting of a sequence of components starting with the root directory of the tree. For example, in the UNIX directory system (probably the most common tree-structured system) the path name */usr/ecc/paper.tex* denotes the file found by starting at the root directory (the leftmost "/"), consulting the directory *usr* to find the directory *ecc*, which in turn contains the entry *paper.tex*. In the UNIX system, only the "/" is interpreted by the directory system; file extensions such as *.tex* are purely convention. Other directory systems provide more support for, and often more restrictions on, the use of file extensions. Another feature of directory systems that is missing from UNIX is the provision of multiple versions of files. Versions are typically specified through additional file name syntax, and file operations typically use different default versions if none is specified. For example, opening a file for reading would default to the most recent version, while deleting a file would default to the oldest version.

A final component is the protection system, often subsumed by the directory system. For example, the directory system can allow access control lists to be associated with each directory entry, and can provide default access controls through an inheritance mechanism. Note that an access control mechanism presupposes some method of securely identifying people. In a distributed environment, this can be accomplished with an authentication service as outlined above.

12.2. Sharing Files in a Distributed System

The ease with which files can be shared in a distributed system is a good measure of the overall success of the system. Several approaches are possible. The lowest level technique is the disk server. A disk server can be viewed as a multiported disk controller whose I/O bus is the network. This approach requires minimal changes to the operating system of the client machine, since the interface is similar to that of a local disk. The abstraction provided is simply that of virtual disk pages. Although read-only sharing of files is simple with this technique, write sharing poses difficulties.

The disk server's interface is too low-level to implement concurrent write operations properly. For example, there is no way to lock a file or to enforce access controls. Instead, the client operating systems have to negotiate among themselves using a separate protocol.

An intermediate level approach is to provide an abstraction of files with unique IDs. The interface to such a file server can allow individual blocks of files to be read or written, as well as logical operations on the entire file such as locking. File servers of this type are usually accessed via a directory system, which must itself be a shared service.

The highest level approach is to use a complete file and directory server, functionally equivalent to the file and directory system on a client machine. Interfacing is again simple because file operations can be intercepted at a high level and redirected to the remote server.

12.3. Integrating Workstation Disks and File Servers

Another issue that is raised when workstations are networked with file servers is how to use workstation disks most effectively. One successful method, used in the Cedar file system [32], considers all shared files to be immutable (read-only), and uses each workstation file system as a cache for some portion of the globally shared file system. Files are created on the local file system and remain private until they are stored back on the shared file server. From that time on, that version of the file may not be modified, and may be shared by other users (subject to normal protection mechanisms, of course). Guaranteeing consistency is relatively simple; the shared file server must provide atomic creation of a new version of a file.

A different approach is taken by the designers of the Carnegie-Mellon ITC file system [30]. Workstation disks are also used as caches, but shared files are not assumed to be immutable. As a result, cache validation is required, initiated either by the workstation before using a cached file, or by the file server when a shared file is modified.

12.4. Integrating File System Name Spaces

Once file servers are used to permit sharing of files in a network, integration of many file name spaces becomes an issue. The integrated name space should allow a file to be named in the same way from any machine in the network, in order to foster portable programs and minimize confusion when users change workstations.

If the different file servers are at the intermediate or low level described above, integration can be achieved through a single (logically centralized) directory service. A more common case, however, is that existing workstations, mainframes, and file servers all have their own file and directory systems that must be integrated into a single name space. For tree-structured name spaces, two schemes are possible. The first uses a super-root that logically contains the roots of all file systems in the network. Additional syntax is used to refer to the super-root in full pathnames. For example, the file name */../A/usr/lib* might be used to refer to */usr/lib* on machine A from any other machine in the network. Advantages of this scheme are that it is simple to implement and guarantees consistent interpretation of file names anywhere in the network. A disadvantage is that this approach is not transparent since the location of a remote file is reflected in its full path name. This problem can be circumvented through the use of symbolic links, a directory system feature which allows a user to impose an arbitrary view on top of the actual tree structure.

The second scheme allows remote mount points in each local directory tree, so that each directory system may have a different view of the distributed system. The problem with this approach is that consistent interpretation of names must be obtained by convention; it is not enforced by any mechanism. The logic of name interpretation on the local machine is also more complicated. On the other hand, it gives individual machines more control over their view of the name space.

12.5. File Servers versus Database Servers

There is growing agreement among designers of distributed file systems that it is important to distinguish between file system and database system functionality. For example, file servers must support efficient sequential reading of small files and creation of new versions of files, but probably do not need to support large files or synchronized modification of portions of files. Database servers, on the other hand, can be used for transactional updates to shared information and efficient access to large files. Making this distinction allows optimized file server and database server designs, rather than compromised designs stretched to fit both classes of needs.

13. Fault Tolerance

Another area in which distributed systems differ from centralized systems is failure semantics. Partial failures, in which some but not all of the components of a system continue to function, are more common in distributed systems and add to their complexity. Various mechanisms are used in order to cope with this complexity. The book by Anderson and Lee presents a thorough overview of fault tolerance techniques [2].

13.1. Transactions for Reliability

Transactions are used to simplify the construction of reliable distributed programs, ones which do not lose or corrupt data. Transactions were first used in database systems [14], but have since been adopted in operating systems [33] and programming languages [20]. A transaction has three essential properties, each of which must be guaranteed even in the presence of processor and communication failures.

Serializability, the first property, means that the concurrent execution of any number of transactions is equivalent to their serial execution in some order. This property insures that if each transaction transforms a consistent database state into another consistent database state, the overall consistency of the database is preserved when transactions execute concurrently.

The second property is atomicity, which guarantees that a transaction is an all-or-nothing operation; no partial effects of a transaction are ever visible to other transactions. When more than one processor is involved, this requires some form of distributed commit protocol, the most well known of which is two-phase commit [14, 19]. At any time before committing, a transaction may abort, leaving the system state as if the transaction had never been executed. The fact that intermediate effects are not visible to other transactions means that the domino effect (cascaded aborts) cannot occur. When a transaction is aborted, one can be sure that no other transaction, either still running or already committed, could have relied on updates performed by the aborted transaction.

The third property is permanence, which states that once a transaction commits, its effects become permanent. Providing permanence in the presence of failures requires some form of stable storage [19]. This involves writing each logical page of data onto more than one disk and modifying the read and crash recovery operations to take advantage of the redundancy. It is still possible that the copies of the disk page can become corrupted in such a way that the read operation would fail; but by increasing the degree of replication, the probability of such a catastrophic failure can be made arbitrarily small.

Crash recovery mechanisms use stable storage in two ways: for checkpoints and logs. A checkpoint is a snapshot of a consistent state that can be restored after a crash. A log is a record of the events or operations that affect the state of the system; it is replayed after a crash. Checkpoints provide faster crash recovery, while logs are less expensive during normal operation. If a combination of these two schemes is used, the log need only be replayed from the most recent checkpoint, and the time between checkpoints can be used to balance the cost of the normal and recovery modes of operation.

13.2. Nested Transactions

Nested transactions are a generalization of single-level atomic transactions, in order to allow them to mesh properly with the concepts of composition and abstraction supported by programming languages. In this scheme, a transaction consists of a tree of subtransactions, with a single top-level transaction at the root. The intermediate effects of a transaction that has not yet committed are visible only to its descendants in the tree. The effects of a committed subtransaction are visible only to ancestors and siblings in the tree. If a transaction aborts, any uncommitted subtransactions must be aborted, and the effects of any committed subtransactions must be undone. The nested transaction model was chosen for the Argus system at MIT [20].

13.3. Replication for Availability

The availability of a system is the probability that the system will be up (either at a particular time or on average). Replication is used to increase the availability of distributed systems, either through the use of a primary/standby architecture or via a modular redundancy scheme. In a primary/standby scheme, only a single component performs its normal functions; all the other components are on standby in case the primary fails. In a modular redundancy approach, all components perform the same function, and some form of voting on the outputs is used to mask failures.

A classic primary/standby architecture is Tandem's method of process pairs [4]. The processes in a process pair execute on different physical processors. One process is designated as the primary, the other as the standby. Before each request is processed, the primary sends information about its internal state to the standby in the form of a checkpoint. The checkpoint enables the standby to complete the request if the primary fails.

The Isis project at Cornell uses a primary/standby architecture for replicated objects [5]. In each interaction with a replicated object in Isis, one replica plays the role of coordinator, and only it performs the operation. The coordinator then uses a two-phase commit protocol to update the other replicas.

Triple modular and *N*-modular redundancy have long been familiar to designers of fault-tolerant computer systems [2]. In triple modular redundancy, every computation is carried out by each of three processors. The results are then compared, and if at least two agree, that value is used. In the Circus system, replication was integrated with remote procedure call in order to support modular redundancy at the program module level [9].

Gifford's weighted voting scheme uses quorums and version numbers to provide replication transparency for files [13]. In this algorithm, read and write quorums (sets of replicas) are chosen so that any read operation will include the most recently written version. Herlihy extended Gifford's algorithm to handle replicated abstract data types [15]. In Herlihy's approach, constraints on quorum assignments are derived from analysis of the semantics of the abstract data types.

14. Conclusion

Well designed distributed systems should strike appropriate balances between the needs for integration and autonomy, and between the needs for increased performance and increased availability. The itemized points below represent the features that project members recommend for inclusion in the operating system, programming language, and support environment of any future distributed system.

- Message-based kernel
- Transparent network inter-process communication
- Remote procedure call facility
- Group communication integrated with remote procedure call
- Conventional software tools extended for distributed environments
- Lightweight processes
- Distributed file system
- Distributed database system
- UNIX compatibility

References

[1] Mustaque Ahamad and Arthur J. Bernstein.
Multicast Communication in UNIX 4.2BSD.
In *Proceedings of the 5th International Conference on Distributed Computing Systems*, pages 80--87. May, 1985.

[2] T. Anderson and P. A. Lee.
Fault Tolerance: Principles and Practice.
Prentice-Hall, 1981.

[3] Robert V. Baron, Richard F. Rashid, Ellen Siegel, Avadis Tevanian, and Michael W. Young.
MACH-1: A Multiprocessor Oriented Operating System and Environment.
In Arthur Wouk (editor), *New Computing Environments: Parallel, Vector, and Symbolic.* SIAM, 1986.

[4] Joel F. Bartlett.
A NonStop Kernel.
In *Proceedings of the 8th Symposium on Operating Systems Principles*, pages 22--29. December, 1981.
Published as *Operating Systems Review*, 15(5).

[5] Kenneth P. Birman, Thomas A. Joseph, Thomas Raeuchle, and Amr El Abbadi.
Implementing Fault-Tolerant Distributed Objects.
In *Proceedings of the 4th Symposium on Reliability in Distributed Software and Database Systems*, pages 124--133. October, 1984.

[6] Andrew D. Birrell and Bruce Jay Nelson.
Implementing Remote Procedure Calls.
ACM Transactions on Computer Systems 2(1):39--59, February, 1984.

[7] Andrew D. Birrell.
Secure Communication Using Remote Procedure Calls.
ACM Transactions on Computer Systems 3(1):1--14, February, 1985.

[8] David R. Cheriton and Willy Zwaenepoel.
Distributed Process Groups in the V Kernel.
ACM Transactions on Computer Systems 3(2):77--107, May, 1985.

[9] Eric C. Cooper.
Replicated Distributed Programs.
In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 63--78. December, 1985.
Published as *Operating Systems Review*, 19(5).

[10] Annette L. DeSchon.
A Survey of Data Representation Standards.
Technical Report RFC 971, SRI Network Information Center, January, 1986.

[11] Whitfield Diffie and Martin E. Hellman.
Privacy and Authentication: An Introduction to Cryptography.
Proceedings of the IEEE 67(3):397--427, March, 1979.

[12] *Reference Manual for the Ada Programming Language*
United States Department of Defense, 1983.
U.S. Government Printing Office, ANSI/MIL-STD-1815A-1983.

[13] David K. Gifford.
Weighted Voting for Replicated Data.
In *Proceedings of the 7th Symposium on Operating Systems Principles*, pages 150--162. December, 1979.
Published as *Operating Systems Review*, 13(5).

[14] J. N. Gray.
Notes on Data Base Operating Systems.
In R. Bayer and R. M. Graham and G. Seegmueller (editor), *Operating Systems: An Advanced Course*, pages 393--481. Springer-Verlag, 1978.
Volume 60 of *Lecture Notes in Computer Science*.

[15] Maurice Herlihy.
A Quorum-Consensus Replication Method for Abstract Data Types.
ACM Transactions on Computer Systems 4(1):32--53, February, 1986.

[16] C. A. R. Hoare.
Communicating Sequential Processes.
Communications of the ACM 21(8):666--677, August, 1978.

[17] *Reference Model of Open Systems Interconnection*
ISO/TC97/SC16, 1979.
Document N227.

[18] B. W. Lampson and M. Paul and H. J. Siegert (editor).
Lecture Notes in Computer Science. Volume 105: *Distributed Systems---Architecture and Implementation: An Advanced Course*.
Springer-Verlag, 1981.

[19] Butler W. Lampson and Howard E. Sturgis.
Crash Recovery in a Distributed Data Storage System.
Computer Science Laboratory, Xerox PARC.

[20] Barbara Liskov and Robert Scheifler.
Guardians and Actions: Linguistic Support for Robust, Distributed Programs.
ACM Transactions on Programming Languages and Systems 5(3):381--404, July, 1983.

[21] *Data Encryption Standard*
National Bureau of Standards, 1977.
Federal Information Processing Standards Publication 46.

[22] Roger M. Needham and Michael D. Schroeder.
Using Encryption for Authentication in Large Networks of Computers.
Communications of the ACM 21(12):993--999, December, 1978.

[23] Bruce Jay Nelson.
Remote Procedure Call.
PhD thesis, Computer Science Department, Carnegie-Mellon University, May, 1981.
Published as CMU report CMU-CS-81-119 and Xerox PARC report CSL-81-9.

[24] David Notkin, Norm Hutchinson, Jan Sanislo, and Michael Schwartz.
Report on the ACM SIGOPS Workshop on Accomodating Heterogeneity.
Operating Systems Review 20(2):9-24, April, 1986.

[25] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel.
LOCUS: A Network Transparent, High Reliability Distributed System.
In *Proceedings of the 8th Symposium on Operating Systems Principles*, pages 169--177. December, 1981.
Published as *Operating Systems Review*, 15(5).

[26] Jon Postel.
Internet Protocol.
RFC 791, SRI Network Information Center, September, 1981.

[27] Jon Postel.
Transmission Control Protocol.
Technical Report RFC 793, SRI Network Information Center, September, 1981.

[28] Michael L. Powell and Barton P. Miller.
Process Migration in DEMOS/MP.
In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, pages 110--119. October, 1983.
Published as *Operating Systems Review*, 17(5).

[29] Richard F. Rashid and George G. Robertson.
Accent: A Communication Oriented Network Operating System Kernel.
In *Proceedings of the 8th Symposium on Operating Systems Principles*, pages 64--75. December, 1981.
Published as *Operating Systems Review*, 15(5).

[30] M. Satyanarayanan, John H. Howard, David A. Nichols, Robert N. Sidebotham, Alfred Z. Spector, and Michael J. West.
The ITC Distributed File System: Principles and Design.
In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 35--50. December, 1985.
Published as *Operating Systems Review*, 19(5).

[31] Eric Emerson Schmidt.
Controlling Large Software Development in a Distributed Environment.
PhD thesis, Computer Science Division, University of California, Berkeley, December, 1982.
Published as Xerox PARC report CSL-82-7.

[32] Michael D. Schroeder, David K. Gifford, and Roger M. Needham.
A Caching File System for a Programmer's Workstation.
In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 25--34. December, 1985.
Published as *Operating Systems Review*, 19(5).

[33] Alfred Z. Spector, Dean Daniels, Daniel Duchamp, Jeffrey L. Eppinger, and Randy Pausch.
Distributed Transactions for Reliable Systems.
In *Proceedings of the 10th ACM Symposium on Operating Systems Principles*, pages 127--146. December, 1985.
Published as *Operating Systems Review*, 19(5).

[34] Robert F. Sproull and Dan Cohen.
High-Level Protocols.
Proceedings of the IEEE 66(11):1371--1386, November, 1978.

[35] Liba Svobodova.
File Servers for Network-Based Distributed Systems.
ACM Computing Surveys 16(4):353--398, December, 1984.

[36] Andrew S. Tanenbaum.
Computer Networks.
Prentice-Hall, 1981.

[37] Andrew S. Tanenbaum.
Network Protocols.
ACM Computing Surveys 13(4):453–489, December, 1981.

AD-A181 853

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNLIMITED, UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
NONE

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
UNCLASSIFIED, UNLIMITED, DTIC, NTIS

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
CMU/SEI-87-TR-5

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ESD-TR-87-10~~8~~6

6a. NAME OF PERFORMING ORGANIZATION
SOFTWARE ENGINEERING INST.

6b. OFFICE SYMBOL *(If applicable)*
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI JOINT PROGRAM OFFICE

6c. ADDRESS *(City, State and ZIP Code)*
CARENGIE-MELLON UNIVERSITY
PITTSBURGH, PA 15213

7b. ADDRESS *(City, State and ZIP Code)*
ESD/XRS1
HANSCOM AIR FORCE BASE
HANSCOM, MA 01731

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
SEI JPO

8b. OFFICE SYMBOL *(If applicable)*
ESD/XRS1

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F19628 85 0003

8c. ADDRESS *(City, State and ZIP Code)*
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
63752F	N/A	N/A	N/A

11. TITLE *(Include Security Classification)*
DISTRIBUTED SYSTEMS TECHNOLGY SURVEY

12. PERSONAL AUTHOR(S)
ERIC COOPER

13a. TYPE OF REPORT
FINAL

13b. TIME COVERED
FROM ... TO ...

14. DATE OF REPORT *(Yr., Mo., Day)*
MARCH 87

15. PAGE COUNT
20

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

THIS REPORT IS ONE OF A SERIES OF SURVEY REPORTS. IT IS NOT INTENDED TO PROVIDE AN EXHAUSTIVE DISCUSSION OF TOPICS PERTINENT TO THE AREA OF DISTRIBUTED SYSTEMS TECHNOLOGY. RATHER, IT IS INTENDED AS AN INFORMATIVE REVIEW OF THE TECHNOLOGY SURVEYED. THESE SURVEYS WERE CONDUCTED IN LATE 1985 AND EARLY 1986.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☒ DTIC USERS ☒

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED, UNLIMITED, DTIC, NTIS

22a. NAME OF RESPONSIBLE INDIVIDUAL
KARL H. SHINGLER

22b. TELEPHONE NUMBER *(Include Area Code)*
412 268-7630

22c. OFFICE SYMBOL
SEI JPO

EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

END

7-87

DTIC