1985

Identifying mechanisms (naming) in distributed systems: goals, implications and overall influence on performance

Wilfredo Rodriguez

Rochester Institute of Technology
School of Computer Science and Technology

Identifying Mechanisms (Naming) in Distributed Systems: Goals, Implications and Overall Influence on Performance

A Thesis submitted in partial fulfillment of
Master of Science in Computer Science Degree Program

By: Wilfredo Rodriguez

Approved By:

Professor James Heliotis (Advisor) 3/28/85

Professor Margaret Reek 3/25/85

Professor Kenneth Reek

Date: March 25, 1985

I Wilfredo Rodriguez prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

Calle Almendro CQ-4
Valle Arriba Heights
Carolina, Puerto Rico 00630

Date: March 25, 1985

TABLE OF CONTENT

1. Introduction

This thesis presents concepts which relate to identification mechanisms in a distributed system. These concepts include goals of the identification mechanism, the implications and influence in the overall performance of the distributed system, and the interaction and relation with other mechanisms. It also studies and analyzes the identification mechanism implemented in several distributed systems, and points out the benefits, drawbacks, and tradeoffs that may exist in each of them when used in different configurations. The conclusions developed may aid in the design of improved identification mechanisms and distributed systems.

Chapter 1 presents basic concepts of naming in distributed systems. The basic components of an identification mechanism are defined. Also several kinds of identifiers and the different levels that may exist in a system are discussed. General goals of distributed systems and the influence that the identifying mechanism has over them are described.

In chapter 2, general identifier structures and identifying methods are analyzed. Their characteristics, advantages, drawbacks and tradeoffs are presented.

Chapter 3 presents identifiers viewed from two perspectives: that of the user and that of the machine (processor). User-oriented identifiers include a discussion of name

servers. Several issues and alternatives used in the design of name servers are presented. The discussion of machine-oriented identifiers includes the analysis of two models of identifiers: server/identifier and UIDs used as local identifiers. Also, different aspects of naming (e.g. name space size, binding of identifiers etc.) that affect the overall performance of the system are discussed.

Chapter 4 presents the analysis of the identification mechanisms implemented in various distributed systems. The identification mechanisms implemented in three network-wide operating systems are analyzed; these systems are: TRIX[Ward80], LOCUS[Pope81,Walk83], and ROSCOE[Solo78,Solo79].

In addition the identification mechanism of a distributed database system named R*[Ceri84,Dani82,Date83] is analyzed. R* implements a naming scheme that has many of the characteristics that contribute to the achievement of the goals discussed below.

Chapter 5 presents a series of generalized conclusions about naming in distributed systems.

1.1. Basic Concepts

1.1.1. Network-wide Operating Systems

Distributed systems are a natural outcome of the development of computer networks. This type of system is characterized by the geographical dispersal of its physical and logical components. By means of effective communication facilities, these components can exchange information and process data in a cooperative manner. Adequate hardware and communication technology that make distributed systems feasible actually exists. The other major component which is the object of much research and development is the system software, that is, the network-wide operating system.

Network-wide operating systems can be categorized into two major groups: network operating systems and distributed operating systems. With a network operating system each host runs its own operating system and the networking functions are performed by a series of programs that interface with the local operating system. A distributed operating system is implemented as a single homogeneous kernel that runs on each host. Each implementation has its benefits and faults, for different configurations and environments. The major functions of a network-wide operating system are similar to those of a local operating system. These include job scheduling and storage management.

In general, the system software has to control resources and provide to the end-user a uniform interface to

these resources, despite the problems caused by physical dispersion and possible heterogeneity. The implementation of real and abstract objects (resources) can be transparent to the user, that is, the user will see no difference when a resource is local or remote.

1.1.2. Identification Mechanisms

An identification mechanism is an essential part of a computer system. It consists of four components: a symbol alphabet, syntax rules, the mapping/context mechanism, and the updating mechanism. Identifiers are constructed using the established symbol alphabet. The identifiers are symbols that designate or reference abstract or real objects (e.g. files, processes, processors, etc.). The identification mechanism uses the syntax rules to define and determine the structure of the identifiers. Eventually, if an identified object is to be manipulated or accessed, the identifier must be mapped using an appropriate mapping function and context, to other identifiers and ultimately to the object itself. Several methods to accomplish this mapping exist (these are discussed in chapter 2). When objects are to be created, destroyed, relocated, or shared, a mechanism for updating the appropriate context is needed.

An identification mechanism can be used for locating and sharing resources, protection, error control, and to

build more complex objects out of simpler ones. The choice of an identification mechanism and its interactions with other parts of the system will affect how well the distributed meets its goals.

1.1.2.1. Identifiers: Levels and Types

The name of the resource indicates what we seek, an address indicates where it is, and a route tells us how to get there[1]. Within the architecture of a system there may exist various different levels and types of identifiers, such as: local addresses, network-wide addresses, local routing information, path names, unique global identifiers and mnemonic identifiers. Each level and type of identifier is different and has its own properties.

Local addresses are those meaningful to a particular host. These addresses are used to access physical implementations of objects. Network wide addresses are those known to the entire distributed system. Generally, they are used to access specific processes (e.g. a process that constantly listens in order to establish a logical connection) that are design to provide specific functions within the system. Routing information is used to transport messages among the

---

[1] Richard Watson, Distributed Systems Architecture and Implementation (New York: Springer-Verlay,1981) p192

communication facilities that lie under the distributed system. The routing information can either be static or it can be dynamically adjust to changes in the topology of the system.

Path names are those used similarly to a hierarchical identifier. By parsing the path name, the system will search through a sequence of directories that eventually will locate the desired file. Pathnames can also be used to locate other objects (e.g. a process) within the system. Usually the beginning of the search is done through a root directory known to every host on the system. Unique global identifiers are those that unambiguously designate an object throughout the entire distributed system. [Leac82] presents an identifying mechanism that uses global identifiers as local identifiers also; the advantages and disadvantages of the scheme are presented. Mnemonic identifiers are used for human convenience and must be converted to machine identifiers at a lower level.

Even though these different levels of identifiers may seem to overlap to some degree, it is important to appreciate their basic differences of purpose in order to avoid confusion.

1.1.2.1.1. Port Names

A main aspect of distributed systems is the interprocess communication service in which the exchange of data is provided via a specific protocol. Processes on different hosts require a unique identifier that will identify the session that they maintain. In addition, it is desirable to multiplex several logical connections to a single physical connection in order to utilize the resources efficiently. Port names serve these requirements. Each port to port association represents a logical connection. Additionally, a process can use several port names in order to support several independent connections simultaneously.

Port names provide a way to obtain a homogeneous name space in a heterogeneous environment The identifiers created by the allocation and mapping method can be defined as port names.

Different algorithms are used to generate port names and relate them to a specific computer system. For example, sequential integers generated by the system can be used as port names. Another scheme used is the concatenation of the host's global identifier with a local integer.

Ports are used to identify resources and their users. They constitute a convenient network-wide naming convention and provide a way to integrate existing naming scheme into a common name space [Pouz82]. Each host is responsible of

binding the identifiers of its local processes into ports.

1.1.2.1.2. Group Names

In large distributed systems it is possible that the identifier space can become very large and therefore adversely affect performance. A counter measure used is group names. With group names the name space is partitioned into a hierarchy of subsets, thus reducing the size of routing tables and the switching overhead. When communication is established within the one group, "short names" are used. On the other hand, if communication is done between objects from different groups a "full length name" is required. An analogous scheme is the use of telephone numbers. Local calls only require part of the number while an overseas call requires a full length number (area code, etc.). An inherent characteristic of group names is that various identifier formats are used depending on the number of partition levels (e.g. network, region within the network, etc.).

1.1.2.1.3. Association Names

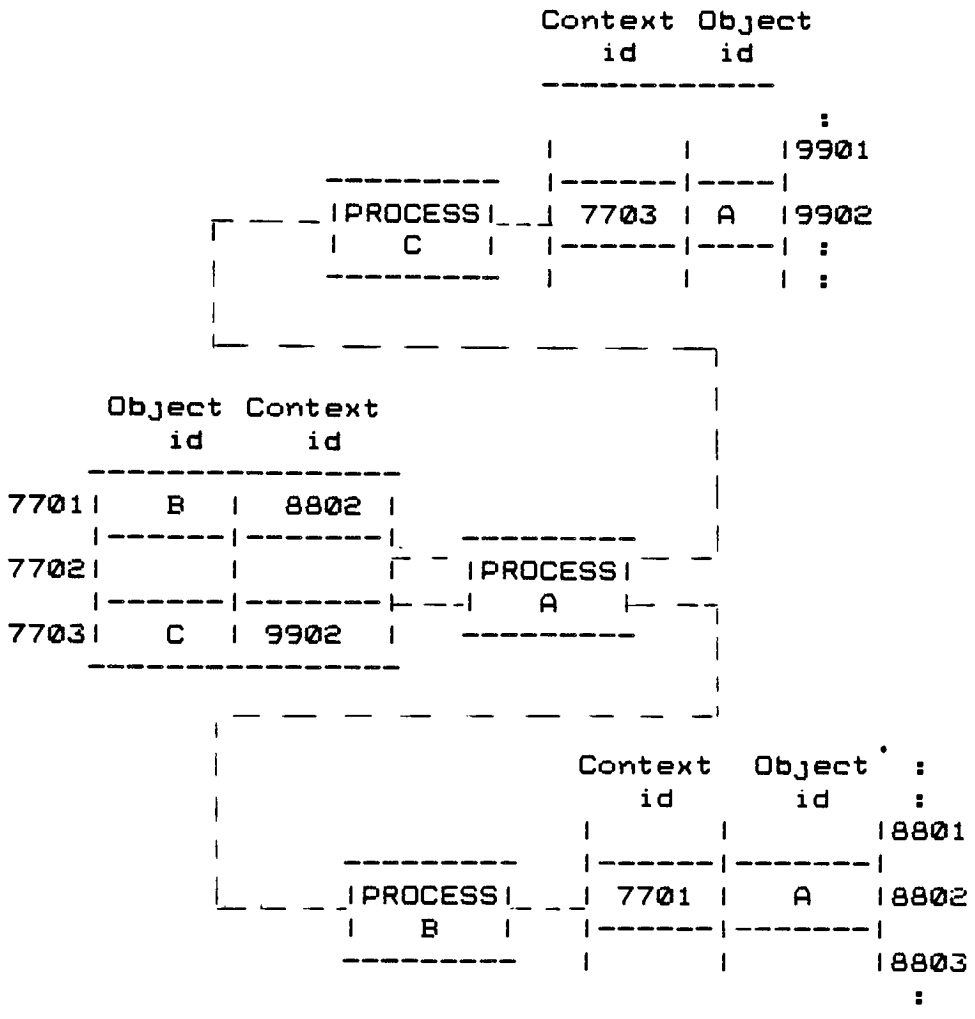
It is possible to exchange messages in a distributed system without explicitly using the addresses of the objects involved. When two processes exchange information they make up an association[Zimm78]. Associations are identified by

context identifiers.

Context identifiers are needed because any two processes can exchange information at any time and it is necessary to correctly interpret the information that arrives at a host. This is done by relating the data to an appropriate context.

In order to relate an arriving message to its context the message must contain a context identifier. The context identifier unambiguously designates a specific context. It could be possible for processes to establish multiple concurrent associations; in this situation the context identifiers are exchange in the initial setup procedure (see figure 1.1). Context identifiers can be seen as an extension to the identifier of a process for multiplexing purposes[Zimm78].

Association names have some advantages. First, the address carried by a message can be shorter than the address of the destination process. Second, security is enhanced because the destination (and source) process cannot be explicitly identified from the message. On the other hand, association names are dynamic. Thus additional complexity is introduced because of the need to synchronize allocation and release of names. Problems can be caused by unreliable communication media (i.e. vulnerable to crashes) that can produce inconsistent status among associated processes.



In this illustration, PROCESS-A has two simultaneous associations, one with PROCESS-B and another with PROCESS-C. The association context identifier of PROCESS-A (with PROCESS-B) is 7701. Thus any message sent by PROCESS-B must contain 7701 as the context identifier. The association context identifier of PROCESS-B (with PROCESS-A) is 8802.

FIGURE 1.1

1.2. Influence of Identifying Mechanism on the Network's Overall Performance

A general and common goal of all computer network, is to provide access to the resources available on the network to every end-user, independent of geographical location, in a rapid and efficient manner. Efficiency implies an acceptable tradeoff between low transmission delay and maximum throughput. Another possible goal is evolution of the network, that is, its capacity to grow by the addition of new hosts. It is obvious that many factors will influence these and other possible goals of a network system. However, the implications and influence that an identification mechanism may have over these goals may not be so obvious.

An identification mechanism has various objectives that can support or obstruct the overall goals of a computer network. An identification mechanism should support at least two levels of identifiers, one that is human oriented and another that is machine oriented. Human oriented identifiers have readable, mnemonic values that make them clear and understandable to the user while machine oriented identifiers are intended to be easily manipulated and stored by host computers.

Identifiers should be generated in a distributed manner to achieve reliability and efficiency. This objective presents a problem with the uniqueness of identified objects and requires some sort of predetermined rules (such as establishing the maximum length of identifiers) to generate the identifiers.

If a network system is to evolve, the identification mechanism must provide for the addition and relocation of objects. This characteristic contains several implications. First, there is the need for dynamic binding between names (identifiers) and objects (resources). Second, a mechanism is needed to update the appropriate context used for mapping between the identifier and the object after its relocation or addition.

The identifier mechanism should support the use of several copies of the same object. This is an important aspect of distributed systems that can help to determine the consistency and reliability of the system.

It is desirable to have multiple logically equivalent servers for a particular resource type in a network system. The identification mechanism must allow for a single identifier at one level to be dynamically bound to more than one address at a lower level.

Finally, the ultimate goal of the identification mechanism is to provide to the end-user a global space of

identified objects that can be accessed and manipulated according to the user's needs. It is clear that the identification mechanism will affect the overall performance of the network system. Other characteristics and design goals in addition to the above described can exist in different mechanisms depending on the architecture and implementation of a specific system. It is possible that an identification mechanism may not have all the desired characteristics discussed above. Identification mechanisms are designed and implemented to meet specific users' needs and depend largely on specific applications.

2. Name Structures and General Identifying Methods

2.1. Introduction

Distributed systems need addresses for the various devices and/or user processes that are interconnected or intend to communicate among each other. A user wanting to contact another user must state that user's address in the initial setup. The situation is analogous to dialing a telephone number to communicate with a person. The telephone system will establish a connection by using the information embedded in the telephone number.

The structure of identifiers will influence the identifying mechanism. There are three general identifier structures that can be used to create identifiers: absolute, relative and hierarchical. Each of them have advantages and disadvantages; tradeoffs exist between them.

Various generalized methods are used to unambiguously identify objects in distributed systems: the hierarchical concatenation method, the allocation method and the mapping method.

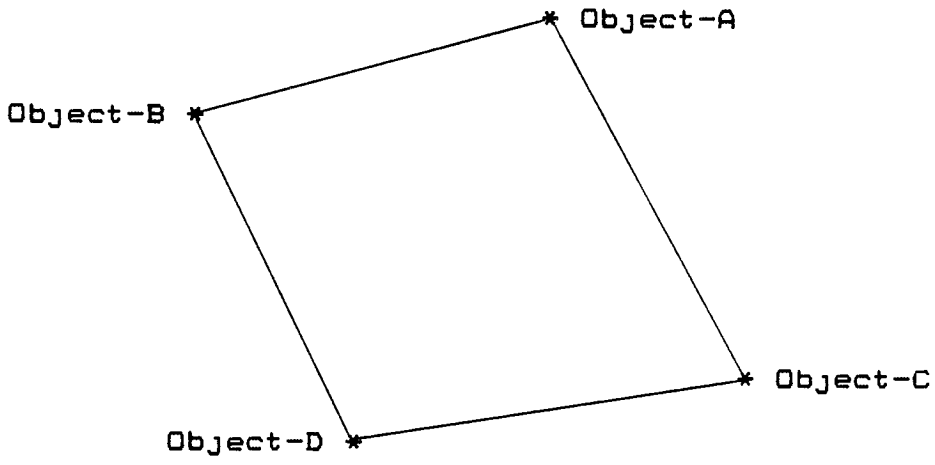
2.2. Naming Structures

2.2.1. Absolute Naming (Flat Name) Structure

When using the absolute name structure, each object is assigned a unique, unambiguous identifier (see figure 2.1). An object can be relocated without altering its name. Routing may be less efficient because the lookup process (resolution of the name) will require a longer search due to larger context tables. Absolute names encourage a centralized naming authority because there is only one mapping function for the entire system. The social security identification number system is an example of absolute names.

2.2.2. Relative Naming Structure

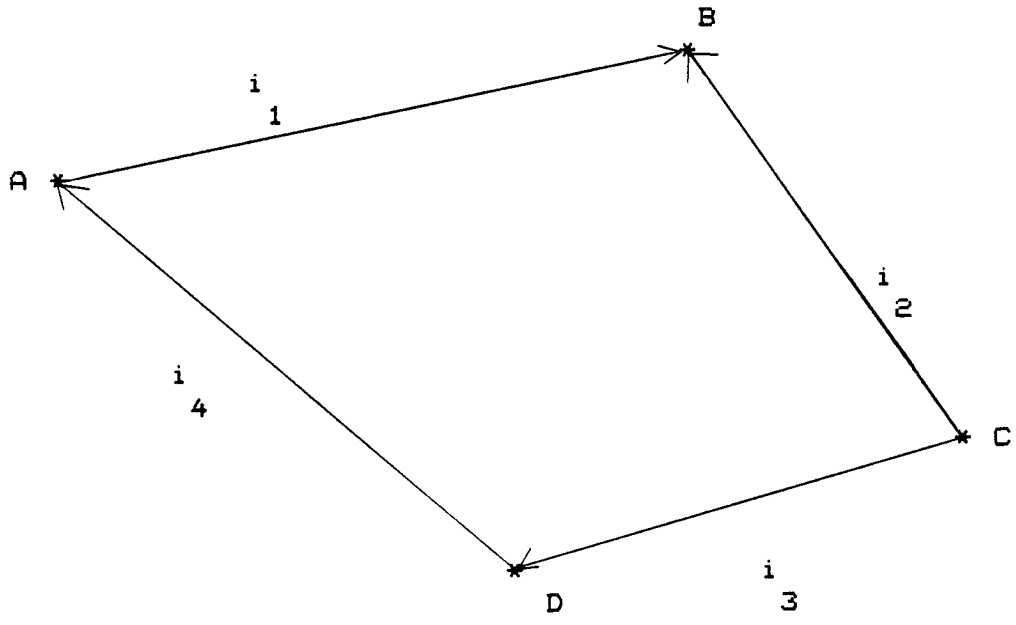
When using the relative name structure, the access path from a source object to the destination object is used as the identifier of the destination object. It is implied that every access path must be unique (see figure 2.2). Within this structure names may be ambiguous. Relative names are unambiguous when they are qualified by the access path. This structure of names encourages decentralization because names are relative to each host. The local database used by the naming authority will be smaller than the one needed in the absolute name structure and the resolution of an object's name will be potentially faster because the domain of a relative name normally is much smaller (the domain of an absolute name is all the names in the system).



An absolute naming structure can be illustrated as a graph in which each vertex, representing an object in the distributed system, has an unambiguous, distinguishing name.

FIGURE 2.1

If names are to be passed among objects, then each host is required to have knowledge of the naming conventions of the others, that is, more coordination is required to share and pass names. This is in contrast to the absolute name structure in which the naming convention is the same for all hosts. The name of a person is an example of a relative name because additional information (such as his address) is needed to disambiguate his name.



Relative naming is illustrated with a directed graph in which each edge represents an access path from one object to another; each vertex is an object and each edge is uniquely identified. In this illustration object B relative to A is named i_1 while object B relative to C is named i_2.

FIGURE 2.2

2.2.3. Hierarchical Naming Structure

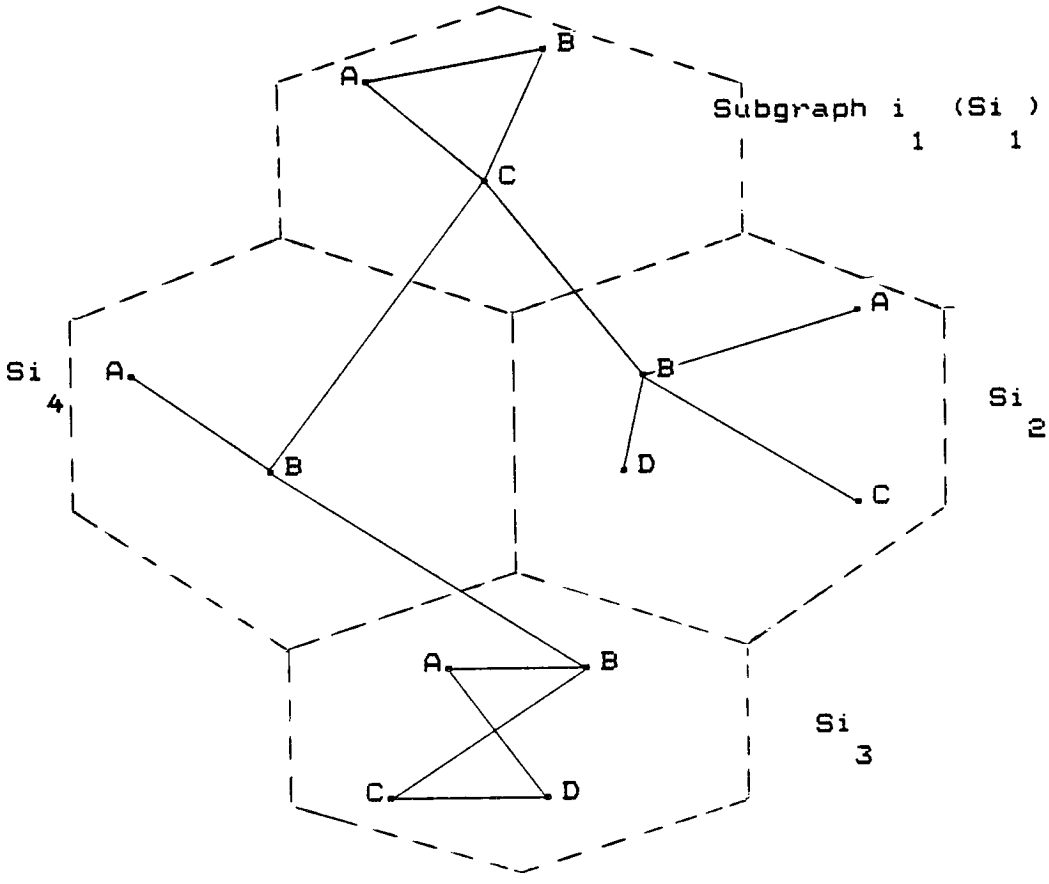
When using a hierarchical name structure the distributed system is partitioned into sub-areas. Thus, the name

of an object is composed of the name of the sub-area and its name within the sub-area (e.g. (object's local name@sub-area)). The absolute naming structure is used to name objects within a sub-area, thus providing unambiguous names within each sub-area (see figure 2.3).

Sub-areas can be identified by either using an absolute or relative naming structure. An example of the relative sub-area names is the interface between the transport mail systems of Xerox and Arpanet. A name of an object within the Xerox system could be OBJ-X.XEROX-SUBA1 and outside of the Xerox system it would be OBJ-X@ARPA-SUBA4. The sub-area part of the name changes relative to its source[Oppe81].

Abbreviated names are a natural outcome of the hierarchical naming structure. In the context of a specific sub-area an object can be referred to by its abbreviated name without any ambiguity (e.g. OBJ-X@SUBA-1 can be referred as OBJ-X in the sub-area SUBA-1). Abbreviation is a relative notion and can reduce the size of the names used within each sub-area.

Hierarchical names allow an absolute naming structure within each sub-area, thus a decentralized naming authority (one in each sub-area) can be implemented. In addition a hierarchy name can help suggest the access path to the object being named when location information is included in the fields of the identifier. When using hierarchical names



The hierarchical naming structure is illustrated by partitioning a graph into regions (subgraphs). The name of an object (vertex) is the region name concatenated with its local name.

A@Si A@Si etc..
1 2
B@Si B@Si
1 2
C@Si C@Si
1 2
D@Si
2

FIGURE 2.3

an important issue to consider is how many levels of hierarchy will be used, as this will influence the performance of the system.

If the name is composed of the concatenation of $i_1 @ i_2 @ \ldots i_k$, k can either be fixed or variable. An advantage of a variable k is that the system may evolve easily. For example, if two existing distributed systems are to be interconnected, the names of the existing objects would change, for example from $i_1 @ i_2$ to $i_1 @ i_2 @ i_3$ without having to alter the existing software used to resolve names. The disadvantage of this is that the software manipulating names is more complex because of the variable number of levels. A fixed k implies simpler and possibly more efficient algorithms for the resolution and manipulation of names. Evolution in this method will require alteration to existing software that manipulates names.

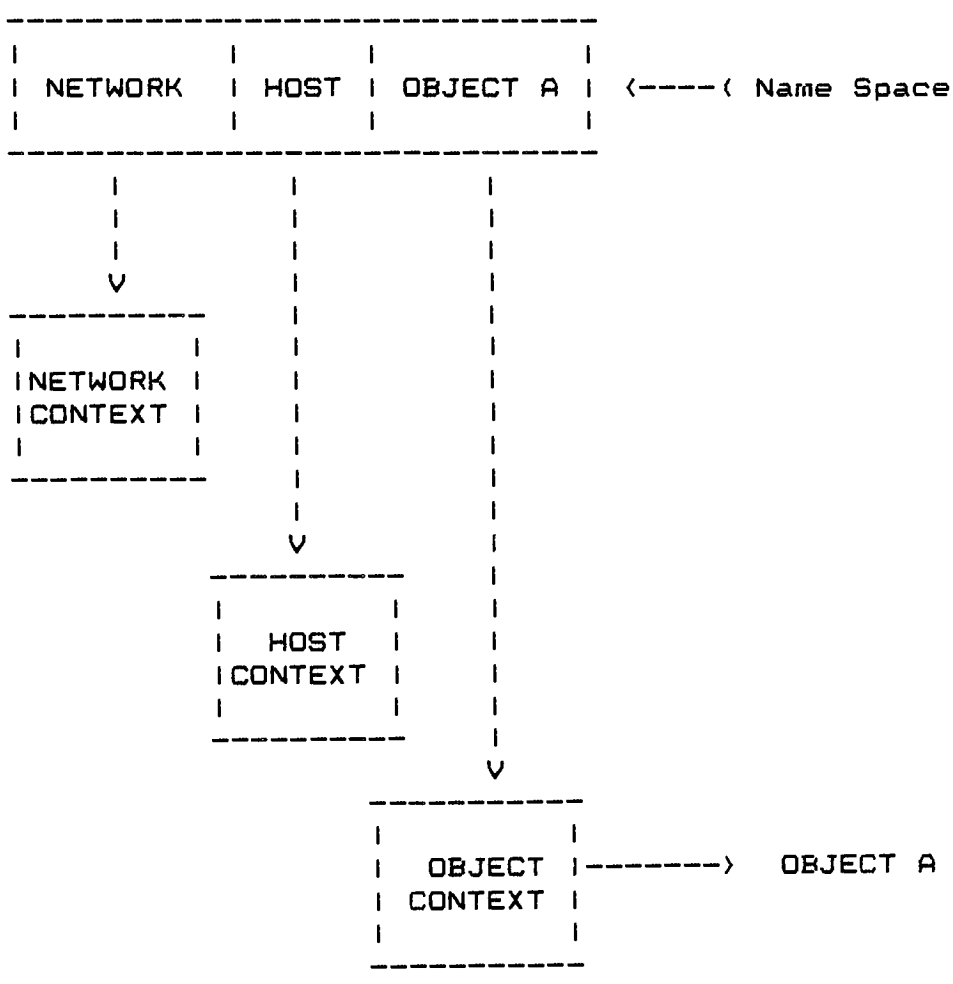
2.3. The Hierarchical Concatenation Method

In the hierarchical concatenation method, identifiers are constructed by concatenating a series of fields that disjointly partition the identifier space. As explicitly mentioned in the name of this method, the structure of identifiers is hierarchical. Some fields are used to indicate

the location of, or the route to, the named object. Other fields identify the specific object within the host's domain. Each field in the identifier may be unique only within the context of another field. An example of hierarchical identifiers used in our daily lives is the routing and transit number used on a check. The first two digits (left to right) represent the federal district, the next two digits represent the zone within that district and the remaining four digits uniquely identify the specific bank on which the check has been drawn. An additional digit is used as checkdigit.

Each host on the network contains a set of local identifiers (names) bounded to its resources. By agreement among all the hosts connected to the network, each host is assigned a unique global identifier. An object's identifier is created by concatenating the unique global identifier with the local identifier assigned by its host (See figure 2.4). An example of a system that utilizes this method is the Resource Sharing Executive System (RSEXEC) [Fors81, Thom73].

The RSEXEC system supports a distributed file system that provides access from each host to files that reside on the other hosts. The identifier of each file is composed of its local identifier and a field that contains its location within the network. To access a file, the operating system parses the identifier to follow through the file system



This figure illustrates an identifier composed of three fields. Each field must be resolved within its specific context. Parsing the identifier will lead to the desired object.

FIGURE 2.4

hierarchy.

The Woodstock File Server[Leac82] is another system that uses the hierarchical concatenation principles. Each file in the system is assigned a "file identifier". File identifiers are composed of the server name concatenated with the local file identifier.

2.3.1. Advantages and Drawbacks

Hierarchical naming is a simple scheme. It provides an easy and explicit way to obtain the location of an object. Also each host can implement a local identifying mechanism that is best suited to its particular environment.

The hierarchical mapping method has several drawbacks. It is desirable be able to have objects relocated without having to find and alter every reference made to them. For example, end-users may want to move dismountable volumes from one node to another or relocate a peripheral from a disabled node to a functioning one. Hierarchical identifiers are inherently tied to the physical location of the object, therefore objects cannot be relocated without altering their identifiers.

There are certain applications in which the location of an object should not be explicitly known (e.g. a military network). The identifiers created with this method reveals explicit information concerning the object's location to a

user with appropriate knowledge of the network topology. This characteristic makes this method inappropriate for this type of application.

Because of the heterogeneity that can exist among the hosts, a wide variety of identifier formats may exist, depending on things such as different physical word sizes, variable length of identifiers, etc. If this situation is severe, parsing of the fields in an identifier will be more complex and less efficient because of the excessive overhead incurred. This also implies that each host needs to know the different characteristics of each other.

2.4. The Allocation Method.

The allocation method associates an identifier to a process in a dynamic way. Each host is assigned a subset of unique global identifiers to be dynamically assigned to processes upon request. Each host contains a process referred as the server process, which is assigned a permanent and network-wide known identifier ([Tane80] refers to this identifier as a "well-known address").

When a connection is desired, (e.g. process X on host-A wants to communicate with process Y on host-B) the initial request is made to the server process. At this point the server process proceeds to allocate and bind one of the glo-

bal identifiers assigned to its host (process Y receives an identifier) and return it to the requesting process (process X). The processes can begin to exchange information now that they are logically connected. Finally, when the communication is to be finished, the process that originated the communication will request the disconnection. The server process will deallocate the identifier that was assigned to the receiving process. This identifier can be reused in a future connection. When identifiers are reused, problems can occur with long-delayed messages. Various techniques exist to handle these problems, but the discussion of them are beyond the scope of this thesis.

2.4.1. Advantages and Drawbacks

An advantage of the allocation method is that it provides the end-user access to remote objects in a transparent manner. It should be recalled that knowledge of the identifiers of the "server processes" and the desired process is required. However, the addresses need not be known. The use of the process server allows access to every object on the system (local or remote) using the same name structure (homogeneous name space).

With the allocation method a single unique identifier can be bound to several generic services. The server process can decide and control which process will implement the

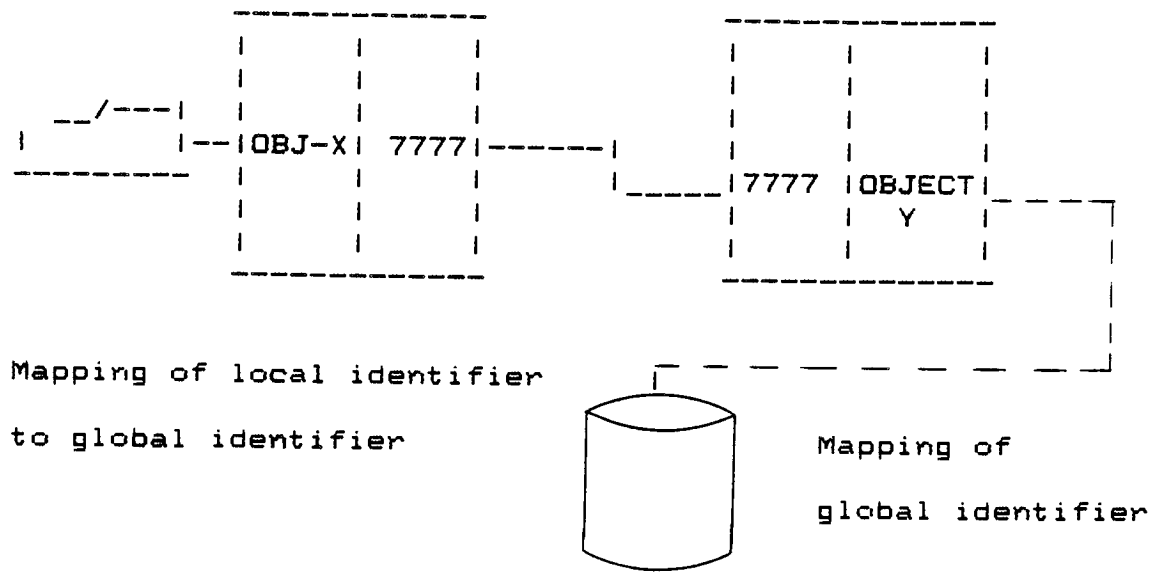
service. This is desirable because the availability of the distributed system is enhanced.

An additional advantage of this method is that only processes that are in execution are assigned identifiers, therefore the overhead involved is reduced.

This method provides dynamic binding between processes manipulating local objects and global identifiers. This property has its cost. The software managing these bindings is more complex than in other mapping methods. Additional mechanisms are required to solve and prevent problems caused by severe transit delays.

2.5. The Mapping Method

In computer systems it became customary long ago to address resources by logical names which are used as indirect pointers to the physical implementations of these resources[Pouz82]. The mapping method statically assigns global identifiers to the local identifiers of the objects that will be accessed by the computer network (see figure 2.5). The mapping method technique is used in many computer subsystems, for example, virtual memory systems and file management systems employ this basic concept. The mapping method is applied in a more global way to distributed systems.



Mapping Method Illustration

1) Host A requests to access object X
2) The local identifier (obj-X) is mapped to global identifier (7777); a centralized naming authority designates the global identifiers.
3) The global identifier is mapped to object Y on host B

FIGURE 2.5

The mapping method is used by the CIGALE packet switching subnet [Pouz82]. In this system the logical destinations are given by name (instead of physical addresses) and the CIGALE nodes maintain tables of all known destinations.

The packets sent through the subnet contain the transport stations logical addresses, not the CIGALE node addresses. With the mapping method, redundant access paths are easily established (when a name is mapped to several paths), thus providing a more reliable subnet. In addition, host connections may be moved from one node to another without service interruption and without changing the name of the host transport station.

2.5.1. Advantages and Drawbacks

The mapping method allows each host to assign its local identifiers in a convenient manner, e.g. each host can use mnemonic symbols. Since each host's local identifiers are not known network wide, there will be no ambiguity when identical local identifiers are used by several hosts.

An object can be relocated without the end-user being aware of it (the subnet system must also implement the mapping method as discussed above in the CIGALE subnet).

This method provides a homogeneous name space for accessing the objects available, despite a heterogeneous network environment. The end-user can request different services available on the network without having to know the names of these services in each of their respective hosts.

Various local identifiers may easily be mapped into a single global identifier, providing the flexibility to refer to an object either by a full or abbreviated name.

There are two major drawbacks to this method. First, if large tables of identifiers exist, the search by a global identifier can consume a large amount of overhead, thus increasing the delay in data transmission. Second, large tables occupy a large amount of memory space ([Zimm78] presents some objections to these drawbacks. Efficient searching algorithms should be used to reduce the search time, also, hardware support could be used if available. Tables can reside on secondary storage as a file directory).

2.6. Concluding Remarks

The general identifier methods discussed above have their advantages and drawbacks. Designers of distributed systems must evaluate the tradeoffs between each method (See Table 2.1). It is possible combine the principles of each method in order to obtain the advantages of each of them.

It is necessary to have an administrative authority in a distributed system. Administrative tasks such as the assignment of identifiers to host and networks must be carried out by a coordinating entity. The administrative entity can be partially or completely automated. A high need

for frequent manual intervention will result in a lower capacity to cope with dynamic changes and higher operational cost.

The administrative entity can be centralized (one for the whole system) or decentralized (one for each region of the system). With a centralized authority uniqueness of identifiers is guaranteed. With a decentralized one, some type of coordination is required to achieve uniqueness.

TABLE 2.1

CHARACTERISTIC	METHODS MAPPING	HIERARCHICAL	ALLOCATION
distributed creation of local identifiers	yes	yes	no
evolution, (support changes in the network topology	yes	yes	yes
map various local identifiers to the same global identifier	yes	no	no
map the same identifier to several objects	yes	no	yes
provide a homogeneous identifier space	yes	yes*	yes
dynamic binding of identifiers to objects	no	no	yes
location transparency (of the object)	yes	no	yes**

* if severe heterogeneity is not present

** the "server process" must have a network-wide known name

3. Machine and User Orientation of Identifiers

3.1. Machine Oriented Identifiers

A machine oriented identifier is one that is used by primitives and the system software to manage objects. Seen from this perspective, emphasis will be given to efficient locating algorithms, identifier structures, protection, synchronization and the type binding used in the system.

The challenge faced by designers of identification mechanisms in distributed systems is to create a globally unique identifier (at some level) that can be resolved within a possibly distributed context and in a potentially heterogeneous environment. Another possible goal is to have the capacity of adding a host to an existing distributed system while still retaining the local identifying scheme (and other exiting software) of that host. The structure of identifiers can either ease or hinder the achievement of these goals. Two different structures of identifiers will be discussed. Several aspects such as the generation of unique identifiers, system architecture requirements, and distributed generation of identifiers will be studied.

3.1.1. Case Studies

3.1.1.1. Server/Identifier Model

A specific structure of identifiers that meets the goals discussed at the beginning of the chapter is: ⟨server-identifier, unique-local-identifier⟩. This structure implies that all objects are managed by a server process and that processes are the fundamental objects that communicate.

This identifier structure provides a uniform global identifier that can partition the global space among local domains. The local identifiers can be chosen as most appropriate and efficient for their local language or operating system environment.

The distributed generation of identifiers requires some sort of predetermined rules for creating them. These rules can be as simple as just limiting the maximum length of the identifier. The network address of the server process (that could be determined by using one of the identification methods discussed in chapter 2) can be used as its identifier and the binding between the server's identifier and its actual implementation is done by routing tables or directories.

3.1.1.2. UIDs as Local Identifiers

Some distributed systems have made use of unique identifiers (UIDs) as local identifiers[Anan83,Leac82]. UIDs can be viewed as large integers or long bit strings.

Several advantages are obtained by using UIDs as local identifiers. First, they provide location independence. Second, UIDs provide an easy way to pass identifiers from object to object and allow objects to refer to other objects by only using their name. Third, additional information about the object can be incorporated in the UID, such as the type of the object and its protection. It is in these ways that the identification mechanism can influence other mechanisms in the system.

An example of a distributed operating system that uses this scheme is Aegis[Leac82]. The structure of an identifier is (host,creation-time,replication). The host bits are used to indicate the host on which the object was created. The creation-time bits are used to ensure uniqueness. The replication bits are used to identify the latest version of the object.

An important aspect to consider when using the UID scheme is the generation of UIDs and the guarantee of their uniqueness. The concatenation of the host identifier (assuming that the host's identifier is absolute) with the current reading from its real time clock is the technique

used in the Aegis O.S. This scheme guarantees the identifier's uniqueness because of a battery operated clock that continues to operate in the event of a host crash and the fact that a host can only create one object during a single tick of the clock.

3.1.2. Size of the address space

An address in its simplest form is a number from a range large enough to encompass all possible objects available on the system. The size of the address space in a distributed system will impact the overhead and flexibility of the system's architecture. If a distributed system is to be transaction oriented, then a large address space (64 bits or more) is desirable because addresses are usually not reused; that is, each incarnation of an object or a logical connection is uniquely identified, and transactions are short-lived.

Large addresses increase the size of the message headers, and consequently the need for memory resources at intermediate and end nodes. They also increase the overhead of transmission. ([Wats81] presents various arguments in favor of the use of large addresses, primarily the reduction in memory cost and the relatively small overall incremental cost to send additional data in a packet).

An important issue of the address space is whether it will be of variable or fixed length. The tradeoff between these two schemes is flexibility, in connecting two networks, versus ease of implementation and less complex software to manage addresses. The IBM System Network Architecture (SNA) uses a fixed address of 16 bits, which provides for 65,536 separate addresses. An example of a variable address size is Digital Equipment Corporation's DECNET, in which network addresses can be any number of bytes. Seven bits of each byte are used as address and the eighth bit is set to 1 if the following byte contains more address information[Mart81]. A general scheme used by other systems is to employ a count field to indicate the number of bits in the address. Variable length addresses allow small network systems to function with low address overhead and large networks to address their resources.

3.1.3. Identifiers and Protection

Protection in distributed systems is a non-trivial matter. Different techniques, such as the use of passwords, authentication, encryption, access lists, capabilities, etc., have been implemented with the sole purpose of providing access to objects only to users and/or to other objects that are entitled to do so. From a machine oriented point of view identification mechanisms and capabilities can be

combined to implement protection in a distributed system.

A capability is an unforgeable token that gives certain rights (e.g. read, write) to the owner. Capabilities can be passed among computing entities, which is a characteristic that makes them appropriate for naming. The Computerized Office System Internetwork Environment (COSIE) Subsystem[Terr83, Terr84] uses capabilities for protection and addressing purposes.

The COSIE subsystem implements mailboxes as a repository for messages, thus allowing asynchronous communication between hosts. In order to send (or receive) a message to (from) a mailbox a process must possess a send (receive) capability from that specific mailbox. In addition, the capability unambiguously identifies a single mailbox. The structure of the capability provides location independence. The level of indirection provided by capabilities allows the system to cope with environmental changes, thus achieving higher availability.

Ports provide a homogeneous name space and logical connections. [Trip83] presents a scheme (The Gutenberg System) in which ports are used to provide protection in addition to their normal use.

The Gutenberg system is a port based, object-oriented system which manages ports, processes, capability directories and the delivery of objects via ports. A process

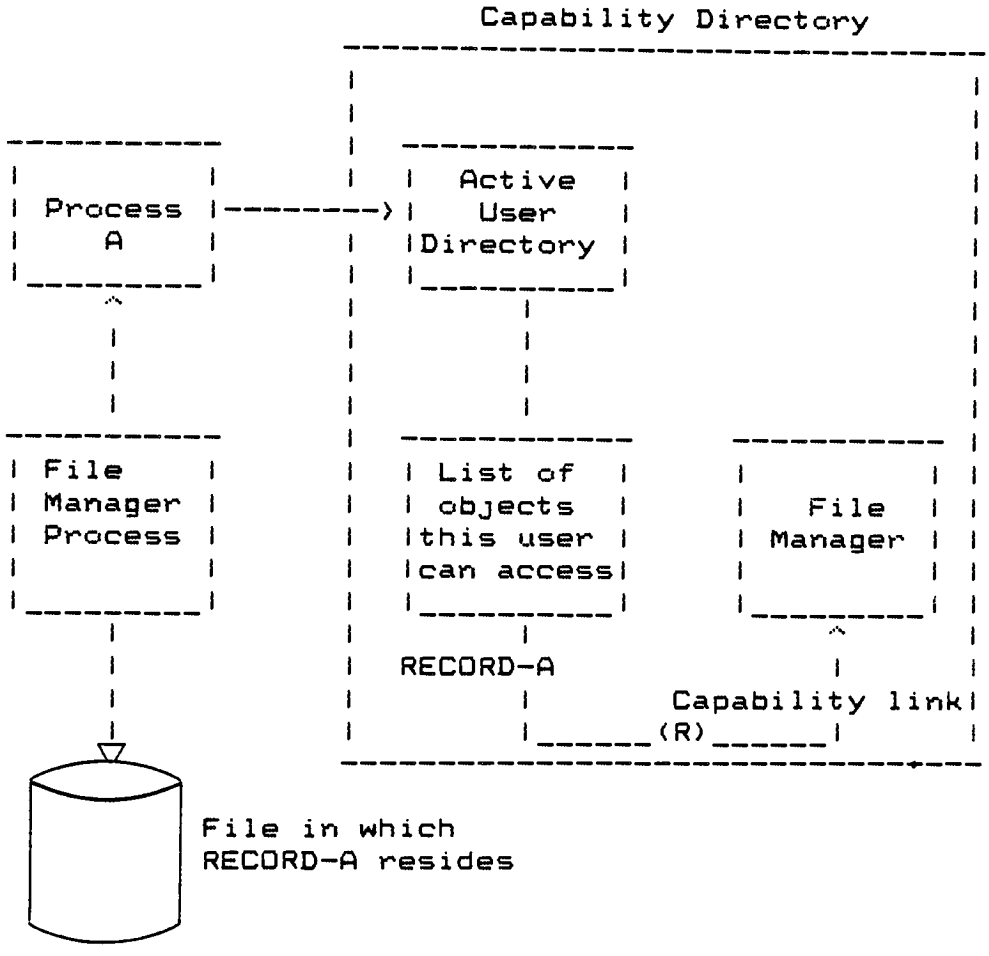
creates a port in order to access an object. The port's type is determined by type of object that is accessed and the operation that is performed via the port. The process executes port operations in a message-based form.

The capability directory is the repository of protection information. Its main function is to restrict the creation and use of ports, consequently controlling access to objects. The basic structure of the capability directory is seen in figure 3.1.

3.1.4. Identifiers, Error Control and Synchronization

When information is being sent over an unreliable media there is a need for some sort of error control. The complexity of the error control mechanism will depend on the degree to which underlying systems (e.g. datagrams, virtual circuits) guarantee error free data.

As a form of error control, units of information (e.g. packets) can be uniquely identified. The receiving entity can keep track of the packets received and discard duplicates. Packets received are acknowledged by returning to the sending entity the identifier of the last one received. The identifiers of the data units can be as simple as sequential integers. In this way, the receiving entity can detect missing or out of sequence units. Several units can be acknowledged when identifiers are assigned sequentially.



This figure presents the basic structure of the capability directory. In this example a user process A wants to access RECORD-A. The active directory is consulted in order to verify access permission. The File-Manager (which is the object that manages the file) is referenced with an R (receive) privilege, thus allowing process A to establish a port connection to read record-A.

FIGURE 3.1

3.1.4.1. Synchronization

The replication of objects in a distributed system increases the availability of resources and provides robustness to the system. Synchronization is needed to provide a consistent state among replicated objects. With the NAMOS[Reed78] system, naming is used to achieve synchronization and provide concurrent access to objects.

The approach to synchronization in NAMOS is based on including the version number of an object as part of its identifier. When an object is updated, instead of rewriting it, a new version is created. The new version of the object will be named by its unique identifier and version number (the time and date of creation can be used as the version number). If a user (or program) wants to access the object, he must indicate the object's name and version. The correct version is obtained by comparing the two version numbers (the user's must be greater than or equal to the object's).

3.1.5. Locating Objects in a Distributed System

The process of locating an object in a distributed system suggests various alternatives, such as not allowing objects to move or to migrate, restricting the location of the object and establishing classes of objects. These alternatives are largely influenced by the structure of the

identifier, whether the location information is centralized or distributed, and on certain restrictions that are imposed on the object's location.

If objects are restricted to staying in the node in which they were created, then the part of the identifier that indicates the creation node can be used as the certain location. This restriction is used in [Feld79], where modules are not relocated. Modules are self-contained objects that can contain data and are capable of processing it (e.g. a program can be composed of several modules). The motivation is that the module then can take full advantage of the hardware and software of the host in which it was compiled. The structure of the identifier used to refer to a module is ⟨computer-number, incarnation-number, site-number, local-module-number⟩, thus uniquely identifying the location of the local module.

Another alternative is to require that objects reside on the same volume (physical disk) as the directory under which they are cataloged. With this scheme a root catalog with a network-wide address is needed (the root catalog is not restricted to any node). In this way objects can be found by starting the search with the root catalog.

There are less severe alternatives to the above restrictions, involving the division of the system into classes. An object could reside on any node (or volume) of

the same class. Various copies of the object could exist within nodes (volumes) of the same class.

If no restriction is imposed, a broadcasting scheme can be used. The request for the location of an object is transmitted to all nodes, and a response containing the address is awaited. This scheme can cause poor performance due to potentially long waits and the overloading of communication lines.

One technique used to improve performance is to maintain a local cache of the addresses of objects. This avoids unnecessary searching once the object has been located. This technique implies the need for a mechanism that will intercept errors that can occur due to relocation of an object after the address has been written in the local cache; the cache is also updated by this mechanism.

The structure of an identifier can contain location information (such as in the hierarchical scheme) thus allowing simpler searching algorithms. On the other hand, if identifiers are location independent then more information (context) is needed. If the information is to be centralized then the system may suffer a reduction in performance, and the risk of not having the locating information available is high. Distributed locating information requires a synchronizing mechanism to update and maintain consistency.

3.1.6. Binding: Static versus Dynamic

Objects are referred to by an identifier. This identifier has to be bound to the physical implementation (or instance) of the object. An important aspect of binding identifiers to objects is the frequency of binding and the set of actions which occur between a binding and subsequent unbinding. Binding of identifiers to objects (or to intermediate identifiers) can be static or dynamic. The main tradeoff is that of performance versus flexibility[Abra80].

When dynamic binding is used, performance is being sacrificed for the ability to adapt to the changes in the distributed system's environment. Performance decreases because each reference to an object requires the resolution of one or more indirect references. With static binding the identifier can contain location and routing information, however, changes in the system's configuration will require a reinitialization or manual intervention in order to correct invalid bindings.

The routing of packets through a computer subnet can be considered as an example of static versus dynamic binding. When fixed binding is used, tables with fixed routes are loaded in each interface message processor (IMP). Changes in the routes require manual changes and reloading of the routing tables. If dynamic binding is used, then an adaptive routing algorithm is employed. The object is the destination

host and the address is the route through which the packet should travel. The binding between the object and the address will vary depending on specific considerations such as congestion and availability of communication lines.

It is desirable to have the performance benefits of static binding and the flexibility of the dynamic binding scheme. This can be accomplished by employing static bindings whenever possible and dynamically rebinding at runtime when bindings need updating.

[Abra80] presents a strategy that combines the benefits of each scheme discussed above. The strategy require a name server to act as a "binding agent". A user that needs the address of an object will present its name to the binding agent, which will return the desired address to the requestor. At this point the user should store the address if future references are to be made to avoid repeated lookups. If a subsequent reference to an object indicates that the object is absent from the present address (which implies a mechanism exist that will efficiently recognize this condition), then the user will present the name to the binding agent for rebinding.

3.2. User-Oriented Names

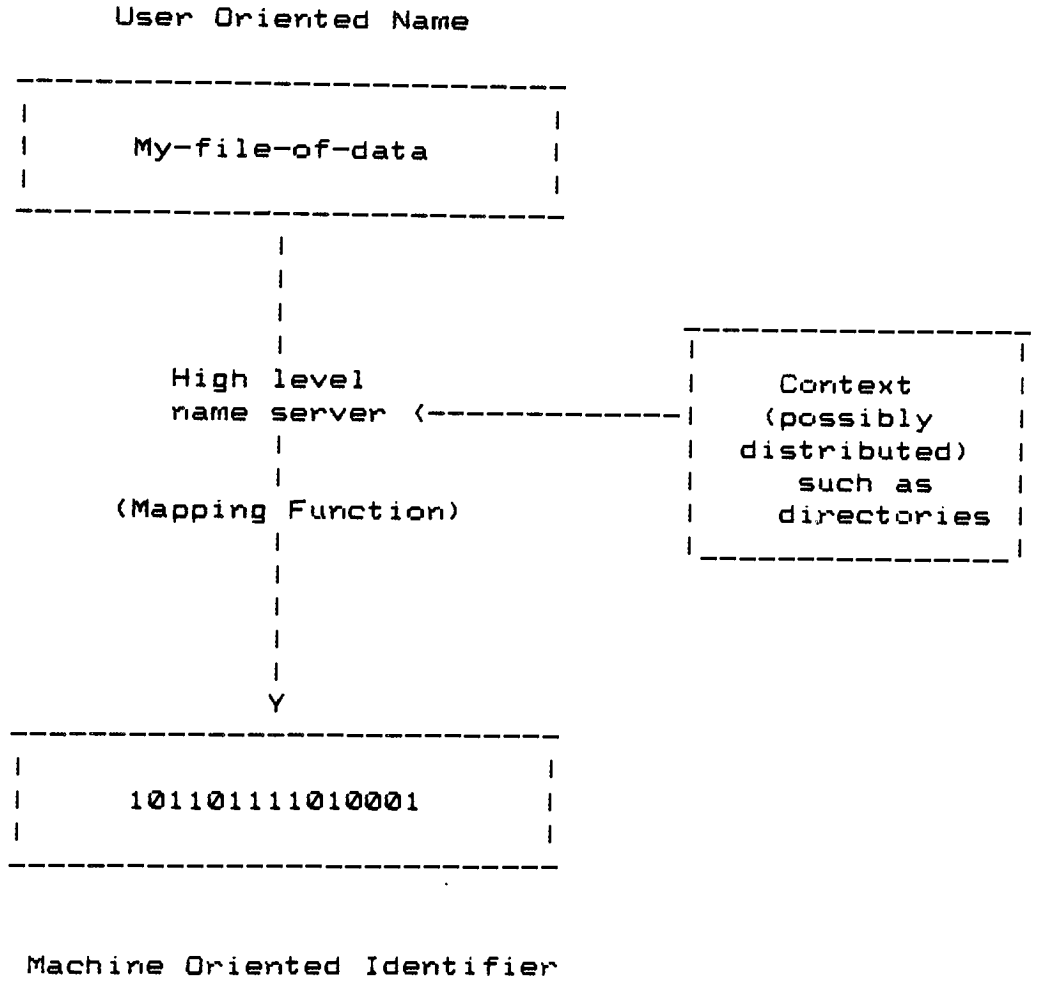
A user-oriented name consists of character strings that are used to ease the access of objects and for the convenience of the user. Higher level naming contexts are required to meet the needs of human users for their own local mnemonic names, and to assist them in organizing, relating, and sharing objects[2]. Humans typically prefer to refer to objects by name while communication is generally done with numerical addresses. Human-oriented names must be mapped to a lower-level machine identifier (see figure 3.2). The mapping of a high-level name to a machine-oriented identifier can be done by a simple and explicit context such as (name, machine address) or it can be done by applying an algorithm to the name in order to obtain the address, e.g. a hashing algorithm. Directories (catalogs) are widely used to support high level naming conventions. They provide a tool for sharing and protecting objects (see figure 3.3). Directories can be centralized or distributed.

A centralized directory is located at a single host. Subsets of the directory can be cached at a local host to improve response time.

If a directory is to be distributed, it can be replicated or partitioned. In the replicated approach each host

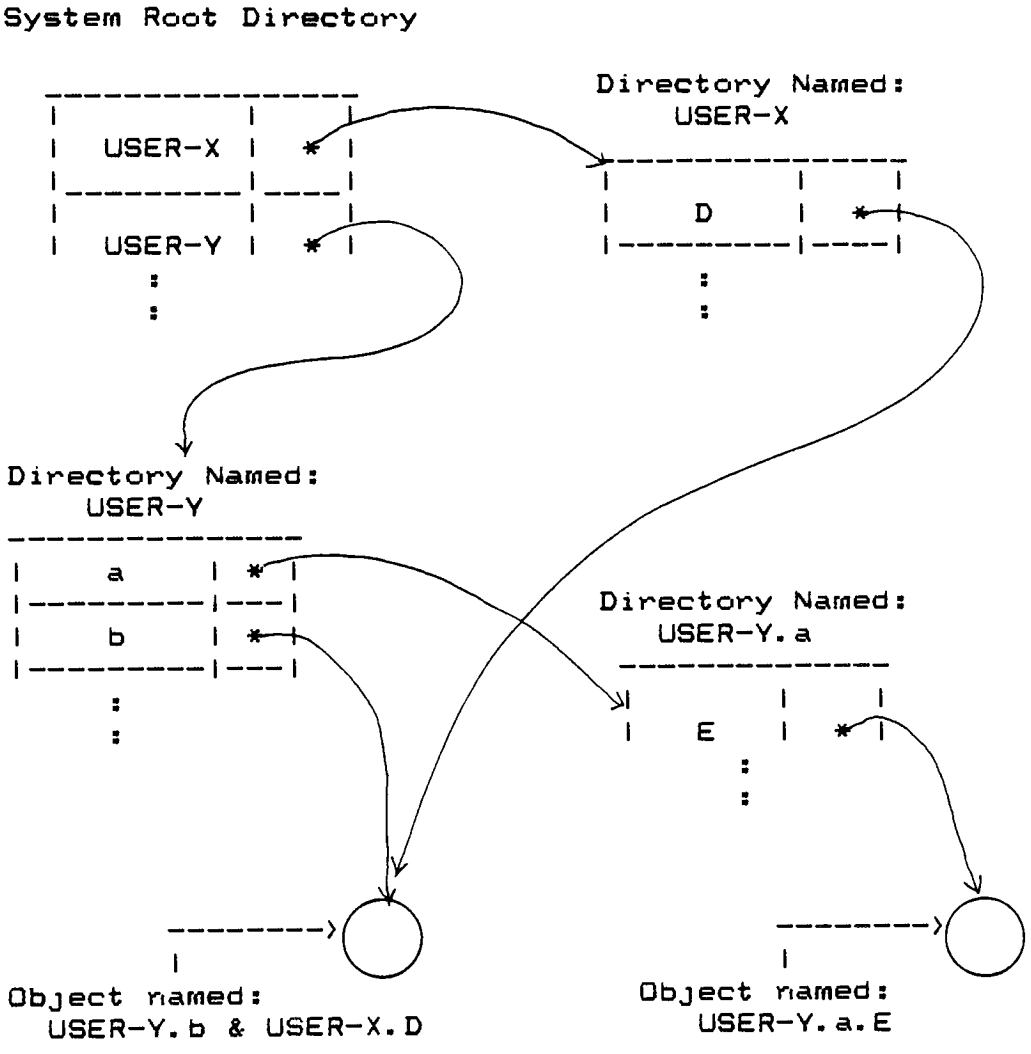
---

[2] Richard Watson, "Identifiers (Naming) in Distributed Systems", New York, Springer Verlay, 1981, p. 203



The mapping between a user-oriented name and a machine oriented name as presented in [Wats81].

FIGURE 3.2



This illustration represents a series of directories used to provide high-level identifiers and to share objects. Protection can be added to objects by including additional information in each entry (e.g. read only, write, passwords, etc.).

FIGURE 3.3

contains a copy of the directory. A mechanism for maintaining consistent directories is required.

In the partitioned approach each host has a directory mapping its own objects. No overlap exists among the directories stored at different hosts; thus the global directory of the system is the logical disjoint union of the physically dispersed directories.

3.2.1. Name Servers

A name server is a utility used to support naming, storage, and retrieval of network-visible objects. By using a name server, a user is able to access a network object by specifying its name instead of its network address.

An analogous relation between distributed systems and name servers is the one between people and the telephone directory. The telephone guide provides its users with a means to locate (find the telephone number of) people by specific names (the white pages) and by generic names (the yellow pages). In the same way, a name server provides to its users the locations (address) of objects by name and can be used to provide generic services (e.g. a pool of printers). Returning to the telephone system, inconsistencies can occur if a person moves to another address or if he changes his name. Similarly, inconsistencies can occur in a

distributed system; the name server is expected to cope with changes in the environment, thus updating its information as soon as possible. The design and implementation of a name server largely depends on the environment in which it is to be used (e.g. distributed office systems, electronic mail service etc.). Various design issues exist; each of them presents several alternatives.

3.2.1.1. Design Issues

An important issue is to decide what type, or structure of names the server is going to support. In addition, the kind of mapping to be supported must be determined. This could be one-to-one (each object having exactly one name), one-to-many (many objects having the same name) or many-to-one (allowing objects to have several names).

Another design issue is, how many servers will there be? There can be one centralized or many decentralized servers. When a network address is mapped to the name that a user (or program) has presented to the server it may be possible to use the returned address as a "hint" rather than being sure that the object is located at that address. If addresses are to be used as "hints" the user must be aware that the returned address may not be correct. It is possible to have a mechanism that can intercept this type of error condition and cause the name to be presented to the

name server for rebinding.

Management issues related with the server must be determined. First, which entity is responsible for allocating names, the name server or the user? The server may or may not support the use of aliases, nicknames and abbreviations. Also, the updating authority must be determined; is the updating process going to be automated (i.e. the server is responsible for self-updating) or is the user responsible for updates?

Protection can be provided by name servers. Who is going to be allowed to access objects which are under the control of the server? Will protection be implemented by the use of passwords, access list, or capabilities?

A possible design issue directly related to performance and flexibility is whether or not to allow the user to bypass the services of the name server, that is, to permit the user (with the appropriate knowledge) to directly access objects via network addresses.

As can be seen, each name server design issue has various options. It is not straightforward to decide which alternative is better than another. Tradeoffs must be considered, and as mentioned above, the environment and application of the server will also influence the decisions. The following section presents a study of two name servers that are designed to meet the demands of their respective

environments.

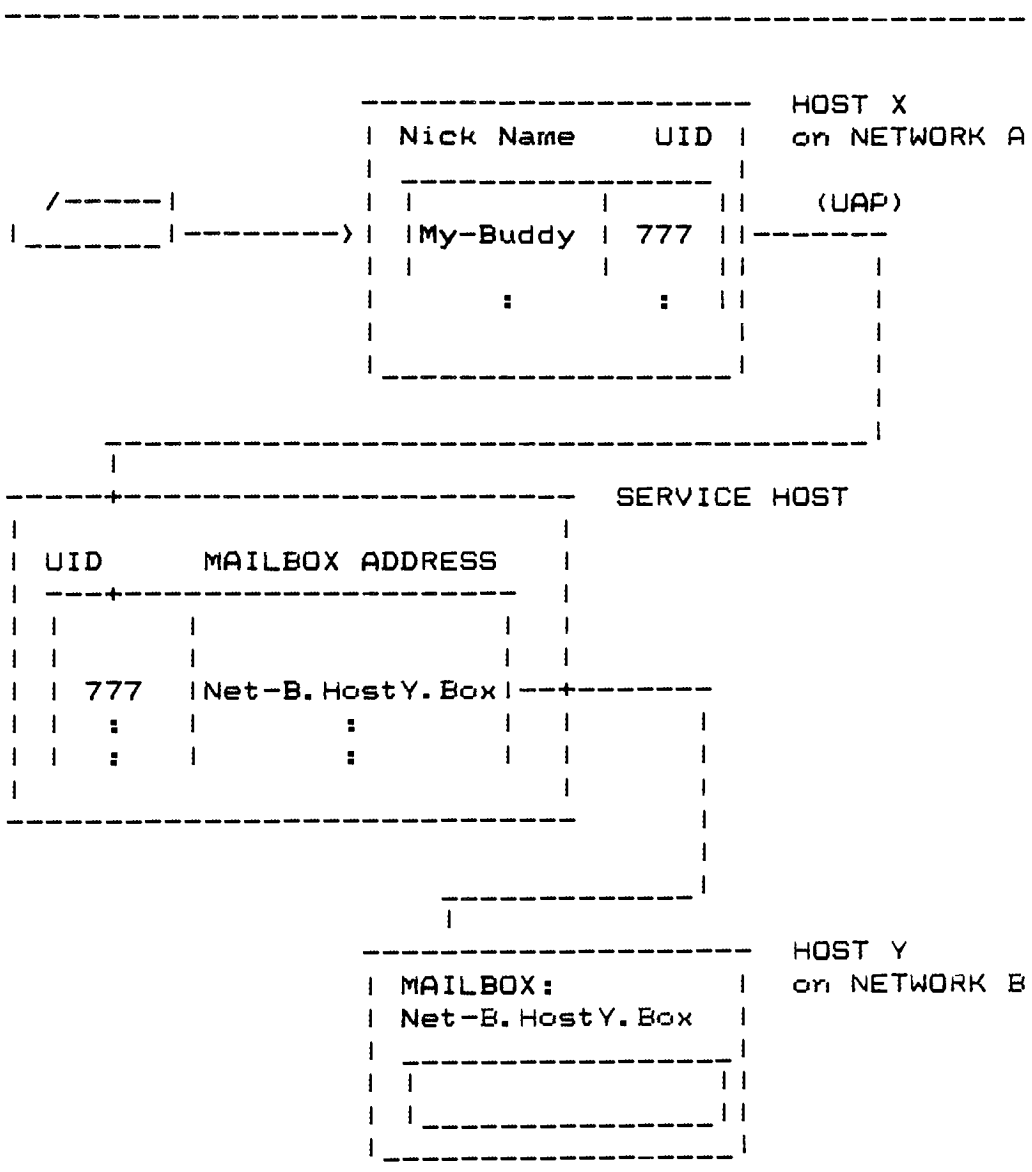
3.2.1.2. Case Studies

3.2.1.2.1. The CSNET Name Server

CSNET's[Land83] main purpose is a computer communications network that provide networks services such as remote login and remote file access to all the computer science research groups in the United States. Problems due to different, and possibly incompatible, naming conventions and address structures may occur when joining computer networks. The CSNET name server was created to simplify communication via electronic mail and to aid in locating available resources.

The internal structure of identifiers used in the CSNET server is absolute. The User Agent Program(UAP) that runs at each host maintains a local mapping table for each local user. The mapping table allows the use of nicknames or abbreviations as illustrated in figure 3.4. Other functions of the UAP are to provide the interface between the user and the system, and to intercept error conditions that may occur.

There is a centralized directory that is stored in the



The mapping is done locally by the User Agent Program to the global unique identifier (UID). On the Service Host machine a centralized directory is used to map the UID to its corresponding mail box.

FIGURE 3.4

Service Host**. The User Agent Program at each host will format queries to the Service Host. The Service Host will use the unique identifier, (sent in the query by the UAP), to lookup the mailbox's network address. The use of a centralized directory ensures that an update need only be done once. Then all subsequent messages for that specific mailbox will be correctly directed. The central directory contains a database entry for each registered CSNET user.

Allocation of names is done by the server and the user, with the help of the UAPs. The UAP allows each user to register himself in the centralized database and maintain the correctness of his database entry (table 3.1 illustrates the fields of each entry). The unique identifier is generated by the system when an entry is added to the database. The user will not be aware of this identifier; it will be used by programs to distinguish accounts from each other and to aid in mail forwarding.

Protection in the CSNET name server is provided by four types of passwords: login password, CSNET password, host password and the super-user password. Access is restricted to the user who owns an entry in the central directory, to administrators at the responsible host and to the Service Host Administrators.

** The Service host is a key component of the CSNET name server. It is the recipient of the database containing all registered users.

TABLE 3.1

Centralized Directory Database Entry Format[Land83].

unique-identifier	key uniquely identifying this entry
account	CSNET account name for entry (user.host.site)
mbox	CSNET electronic mail address of entry owner
csnpass	password for changes to entry from other than home host
fullname	full name of entry owner
address	U.S. Post Office address of entry owner
phone	phone numbers of entry owner
misc	miscellaneous information about entry owner

A login password is required for a user to login to a host connected to the CSNET system. If a user then wishes

to to make an entry to the central directory, he must use the appropriate command with the required parameters. The transaction is encrypted using the host password as an encryption key. The UAP then sends the transaction to the Service Host, which proceeds to decrypt the transaction using that host's password and to make the appropriate entry.

The two-level authentication (the user satisfying the local UAP with his password and the host satisfying the Service Host with its host password) provides the access security to the CSNET name server from the user's home host. If access to the server is to be made from a host other than the user's home host then the user's CSNET password is required. A user registers his CSNET password by updating his entry from his home host. This password is encrypted with the transactions sent by the user to the Service Host. In this way the Service Host can authenticate the user.

The super-user password is used by the Service Host Administrators to modify directory entries and/or install new host entries.

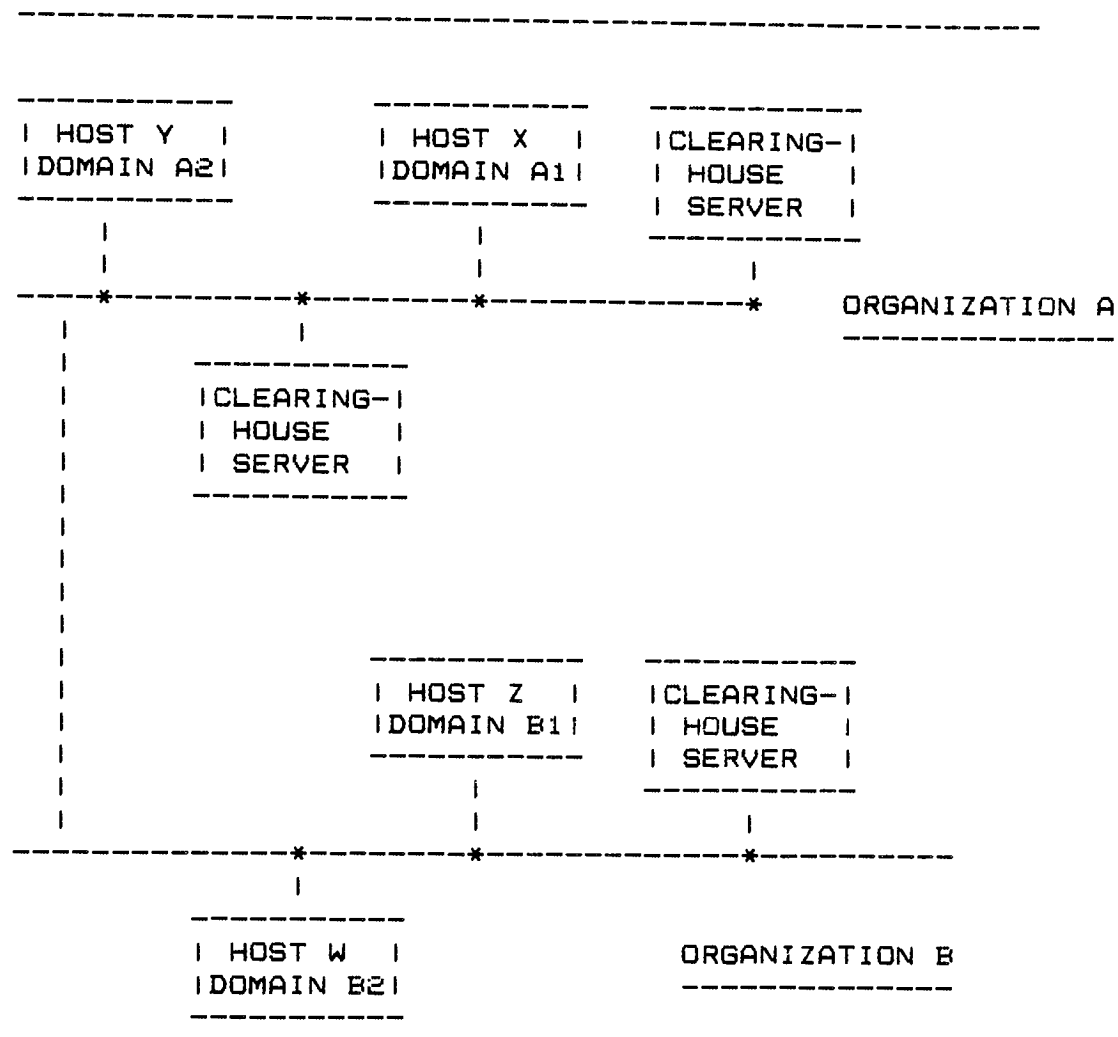
The CSNET name server had a specific design goal: that of allowing a user of the CSNET system to bypass the services offered by the server. Thus users (or programs) are allowed to send mail by specifying the network address of the destination in order to avoid the overhead of the name

resolution and allow access to objects in event of a server failure.

3.2.1.2.2. XEROX Clearinghouse

The Clearinghouse is the binding agent in Xerox Network Systems (including the Xerox 8010 Star Information System) [Oppe81]. It is a main component of the underlying distributed system's architecture. The main objective of the Clearinghouse is to provide name and location information of distributed objects to different interconnected networks used in a distributed office environment. Objects may be individuals such as workstations, file servers, and people or may be groups of other objects such as in a distribution list.

The structure of identifiers used by the Clearinghouse is hierarchical. Every object has a unique identifier that is constructed as follows: (local-name@domain@organization). The division of organizations into domains is logical rather than physical. In addition, each object can have one or more (unique) alias name. The motivation of a three level name is to facilitate the addition of new organizations. Figure 3.5 illustrates a configuration of a system with several Clearinghouses.



This figure illustrates a possible configuration of two organization's local networks. Several servers are scattered throughout the internetwork.

FIGURE 3.5

There are various types of mappings between names and objects. The aliasing supported by the server provides a many-to-one mapping. In addition to the locating of an

object by its name, objects can be located by the use of generic names, thus providing a one-to-one and one-to-many mapping respectively. A name is bound to a set of properties. A property is an organized tuple of information that contains the type and address of an object. Properties are discussed in further detail in Appendix-A.

The Clearinghouse and its associated database are decentralized and replicated to some extent, i.e., an entry of an object can be duplicated among servers. There are many local Clearinghouse servers scattered throughout the internetwork. Each local Clearinghouse stores and manages a copy of a portion of the global database. The total services that are provided by these local Clearinghouse servers will be referred as the Clearinghouse. Decentralization and replication increase efficiency, security and reliability. Access to a physically close server can provide shorter response time. Each organization can control the access to its Clearinghouse server. If a server is not available, it is possible that another server can respond to the request. Updates to the servers are the responsibility of the users; therefore, temporary inconsistencies can arise among local database copies. Addresses returned by the server should be treated as a "hint" to the location of the object (the user must verify that the object is actually there).

The allocation of names is managed by a naming authority (one for each domain) which is responsible for making

sure that different objects have different names. Each domain is responsible for the allocation of its names. Object names are proposed by system administrators and validated by the Clearinghouse of its domain. The Clearinghouse responsible for a specific domain checks that the requestor is a registered system administrator and that the name is not already in use.

Each organization must choose the people to be assigned as system administrators. Their responsibility is to select domain names; by querying the Clearinghouse they check to see that the domain name is not already in use. A central naming authority will validate new organization names by querying the Clearinghouse and authorizing the use of the name.

Protection in the Clearinghouse has three main issues: authentication, access control and interorganization security.

Similar to the CSNET, each user is assigned a password that is needed in order to access the Clearinghouse. Each server has a password that is used when it sends a request to another server. The Clearinghouse maintains a database mapping names of clients and other Clearinghouses into their valid passwords.

After a user has been authenticated, the restrictions on the operations that he can execute are provided by access

control lists. Each domain has an associated access control list. The format of an access list is:

```
{(set-of-names , set-of-operations )}
              1                    1
...
{(set-of-names , set-of-operations )}
              k                    k
```

Each tuple consists of a set of names and the set of operations that each "user" in the set is allowed to execute. As an example,

```
{(User@CS@RIT),{AddMember,DeleteMember,LookupGroup})}
```

would allow User of the domain CS in the organization RIT to execute three operations only.

It is possible for an organization to interconnect its network with another organization's network and still have some degree of protection (or restriction). This is possible by the use of a Clearinghouse Sentry(CS). The CS is a component in the internetwork router joining two networks (similar to a gateway). It acts as a filter between each request and the Clearinghouse server. The CS may reject or accept any request that is directed through it. If the request is accepted, the CS will reformat the request and send it to the corresponding server.

4. Analysis of Implemented Identification Mechanisms

4.1. TRIX SYSTEM

4.1.1. Background and Major Features

TRIX is a kernel operating system designed to be used on a network of interconnected processors. The development and implementation of TRIX was done by the Real Time Systems Group of MIT Laboratory for Computer Science. The name TRIX is derived from the combination of functional characteristics of both the UNIX and MULTICS operating systems[Ward80]. TRIX contains two fundamental elements, streams and processes.

4.1.1.1. Streams

Streams are full duplex communication paths between processes. A distinguishing characteristic of this system is that it binds identifiers to streams rather than to objects. For example, a stream named "date" can provide the current date either by reading a local file that contains the information or by communicating with a remote process. The assignment of identifiers to streams provides a high level of abstraction because the semantics associated with the stream are independent of the mechanism (at the far end of the stream) that implements it.

Underlying TRIX stream communication is a protocol for passing messages between interconnected processes. At this low level, stream communication is done asynchronously, so processes are not blocked. The reason for this type of communication is to allow a process to sustain several active conversations simultaneously. For example, multiplexing a stream into several substreams is possible. This would be done by pairing a concentrator process at one end of the stream with a deconcentrator process at the other end.

Stream communication takes the form of a master-slave relation between the connected processes, which makes them asymmetric despite their full-duplex nature. Each stream has one or more requestor ends and a handler. Each end can be connected respectively to processes that issue requests (such as read or write on the stream) and a process that executes the requests.

4.1.1.2. Processes

Processes manipulate and control objects, and communicate by the use of streams. The communication between different hosts is also done by establishing a stream that connects processes that are specificly designed and dedicated for network functions. Thus, access to a remote process and access to a local one appears to be the same. It is not practical to implement passive data objects (e.g.

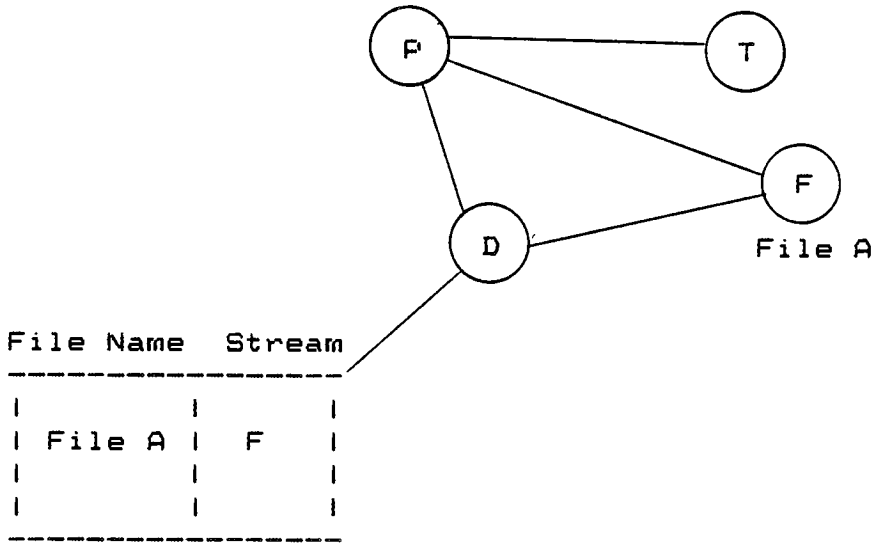
directories) as active objects (a process communicating over a stream) because of the overhead. To overcome such efficiency drawbacks, TRIX provides what are called system-processes. System-processes are semantically the same as ordinary processes but they rely on conventional passive implementation techniques[Ward80].

4.1.1.3. Naming

Each end of a stream is uniquely identified by a stream descriptor token, similar to a file descriptor that associates and contains information about files used by a program (e.g. IBM's file control blocks). Thus, a process must use the stream descriptor of a specific stream in order to request services. For example, if a process wants to write a record of X bytes long from "buffer" it would execute the following command :

write(stream descriptor, Xbytes, buffer).

The write request is forwarded to the process at the handler end of the stream. Figure 4.1 illustrates a simplified configuration of the relation between processes communicating by streams. The role of directories is to bind identifiers to streams.



This figure illustrates a process named P which is reading a file (File A) by being connected to the stream which communicates with the process (F) tnat manages the file. The process D manages the directory that associates names with streams. Thus, directories are the binding agents of the system. Process P is writing to a terminal device by communicating over a stream with process T. A possible sequence of commands is:

```
              :
              :
 file-stream-descriptor := open(File A, directory D)

      while (boolean expression)
          {
           read(file-stream-descriptor, X bytes, buffer)

           write(terminal-stream-descriptor, X bytes, buffer)
          }
             :
```

FIGURE 4.1

4.1.1.4. Directories

A directory in the TRIX system is a process which associates names with streams. Among the operations that can be performed by a directory process are: opening a file, adding names to the directory, deleting names from the directory and listing directory entries. For example, (as was seen in figure 4.1) if a process wants to access a file it must first open the the file by executing the following call:

STREAM-ID = open("name", D).

D is a stream connected to a directory. The directory process would receive the open request through D. Then the process would proceed to lookup the name and return the stream descriptor that communicates with "name".

The "name" of a stream can be a sequence of directories that must be followed in order to communicate with the desired process. For example,

open("dir-x/proc-y", D) = open("proc-y",open("dir-x",D))

thus the resolution of proc-y (i.e. acquiring the stream descriptor) is done by following the directory structure implied in the stream name. In this example "dir-x" is attached to a directory stream and identifies a stream named "proc-y" within the directory "dir-x".

A newly created process communicates with the TRIX kernel by an environment system stream which is inherited at creation time. The system stream has similar function to a

directory stream, in that it associates names such as: standard-input, working-directory etc. to their respective streams.

Processes can conveniently communicate with remote processes by exchanging their system stream descriptors. A user's process that is interacting with a remote service (e.g. a text editor running at another node) might pass its system stream in order to maintain his local naming context within the foreign system.

4.1.2. Discussion of the Identifying Mechanism

4.1.2.1. Structure of Identifiers

The identification mechanism of TRIX is unique; streams receive identifiers. The structure of a stream descriptor is absolute, thus unambiguously identifying the request end of the stream. Pathnames are supported, and provide a way of sharing and efficiently locating a process within the system. An identifier may contain an arbitrary number of components, but a directory is responsible only for the interpretation of the first component. The remainder of the identifier is passed to the subsequent directory named for further interpretation. For example, the name:

X/Y/Z

is resolved by the use of two directories. A directory named

X will return a stream connecting to a directory Y, which will return a stream connecting to Z.

The distribution of name resolution among a sequence of interconnected processes (directory processes) has its advantages. Arbitrary mechanisms may be implemented by an "intelligent" directory. For example, if a file named FILEA is written in ASCII code and a process on a remote host that interprets EBCDIC needs to access it, the pathname of the file could be: Intelli-dir/FILEA. The directory process (Intelli-dir) would convert the file, and the process on the remote host would not be aware of the conversion procedure.

4.1.2.2. Homogeneous Name Space

A homogeneous name space is provided based on:

1) the existence of the TRIX kernel on each processor
2) the use of absolute structure stream descriptors
3) the use of pathnames.

4.1.2.3. Passing Identifiers

In addition to directories, system processes are provided to efficiently interconnect streams. Thus a system process can be used to connect two existing streams or to merge two streams into a third one. The ability to pass stream descriptors among processes via messages, helps allow

processes to be interconnected easily.

4.1.2.4. Mapping

The structure of streams provides various kinds of mappings. Since there can be various requestors at one end of the stream a many-to-one mapping is supported. The handler process of the stream can be used to implement generic services, i.e., one-to-many (however, this is possible based on TRIX's architecture, not on its identification mechanism).

4.1.2.5. Location Transparency

Based on the stream scheme of TRIX, the implementation of services is transparent to the user. Consequently the location of the process that implements the service is also transparent to the user. Access to a remote process is done in the same way as to a local one. In the former case, processes on separate machines support networking functions. The connection to a remote process would require obtaining the stream descriptor of the network process.

4.1.2.6. Protection

The naming scheme of TRIX influences the protection mechanism. Stream descriptors are passed between processes by request and reply messages. The system prevents processes from making copies of stream descriptors. If a process passes a stream descriptor it loses access to the stream. Therefore a process can control the access to

objects by limiting the distribution of their descriptors.

4.1.2.7. Other Remarks

The TRIX system confronts several challenging problems. One of these problems that relates to naming is the management of removable volumes. A process (e.g. a directory or a file) on a removable disk may be connected by a stream to active processes. The dismounting of the device presents the problem of whether or not the remaining connections should be retained. The solution to this problem presented two alternatives. The first one was to retain all the streams connected to dismounted volumes. The garbage collection mechanism of TRIX would have to support an arbitrary large set of references to dismounted devices. The second alternative was the one adopted: to disallow streams to and from dismountable storage volumes. Thus objects (in all hosts) are restricted to be relocated only within a subset of the storage available.

4.2. LOCUS

4.2.1. Background and Major Features

LOCUS[Walk83] is a UNIX compatible distributed operating system developed and implemented at the University of California at Los Angeles (UCLA). It is based on a set of homogeneous systems (VAX/750's) connected by a standard Ethernet. The main objective of LOCUS is to provide network transparency, i.e., to provide to its users the illusion of a single machine. All the resources available on the system are accessed in the same manner, independent of their location. The file system and its distributed naming catalog are essential components of the system. A great part of the services provided by the operating system involve file management. Also, catalogs provide a generalized name service that may be used by other mechanisms. The naming mechanism implemented in LOCUS has two levels: high level identifiers (hierarchical) and system identifiers.

4.2.1.1. Identifiers

File identifiers (high level) have the same structure as a UNIX centralized system, i.e., a hierarchical name space. Each file is accessed by using its pathname, which indicates the series of directories that have to be read before accessing the file. Each element of the path name is

a character string. The format is:

/CHARACTER STRING {/CHARACTER STRING}*

Even though a pathname is used to retrieve the lower level identifier of an object, no notion of the object's location is presented, thus objects can be relocated without altering their pathnames.

Internal identifiers (system identifiers) in LOCUS have the following structure:

(logical-file-group-number, file-descriptor-number(inode))

((LFGN,FDN)). The file group number uniquely identifies a logical volume. The file descriptor number is an index into the array (per group) of file descriptors.

4.2.1.2. File System

The file system in LOCUS is seen as a single tree structured naming hierarchy. The single tree structure includes all the file systems on all hosts. Files are replicated within the logical group in which they reside (but not necessarily on all the physical sites that compose the group). The various copies of a file are assigned the same file descriptor number (inode). The LOCUS system is responsible for maintaining consistency among replicated files and for providing users with the latest version of a file. By the use of the mount** mechanism file groups are logically connected. The LOCUS system manages information of all

mounted file groups (a network wide mount table).

4.2.1.2.1. Directories

All directories are replicated entirely on each host. Better performance and higher availability motivates replication of directories. Directories at high levels, are mostly read rather than updated while lower levels such as user directories tend to be updated more. The root of the tree is usually the starting point for the resolution of a high level identifier.

4.2.1.3. Site Interaction

File operations can involve more than one host. They are executed following established synchronization protocols such as those discussed in [Walk83], [Pope81]). Three logical sites can be involved, the using site (US), the storage site (SS), and the current synchronization site (CSS).

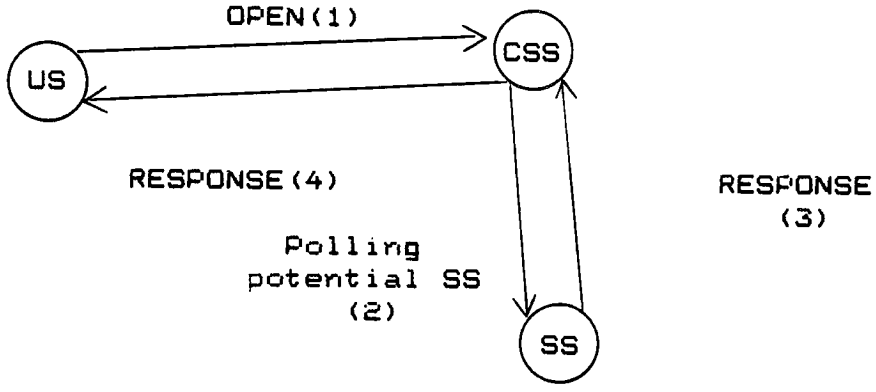
The using site is the one that issues a request (e.g. to open a file) and to which messages are to be sent. The storage site is where a copy of the requested file resides.

** A logical file group that is incorporated into the accessible structure of the system is said to be mounted. The mount command informs the LOCUS system kernel of the existence of the file group.

The current synchronization site enforces a global access synchronization policy for its file group. There is only one CSS per logical file group, but a site can be the CSS of various logical file groups. The CSS selects the SS from which a file is to be accessed. The decision is done based on the information that the CSS contains about which sites store the file and where the most current version of the file is. The following example presents the protocol followed and the interaction of sites.

Example-1: Opening a File

Initially the pathname is resolved in order to obtain the corresponding (file-group-num, file-descriptor-num) ((LFGN, FDN)). The US proceeds to make an "open" request to the CSS. The CSS then polls potential storage sites and selects the appropriate SS, since the CSS has the lastest version number of the file. When an SS is polled it compares its version number against the one received. After the SS has been selected, the CSS proceeds to send the information needed (e.g. file size, permissions etc.) to the US. Figure 4.2 illustrates the sequence of steps followed as presented in [Walk83]. It is possible for a site to execute one or more of the functions discussed, (e.g. the CSS can be the SS). LOCUS provides a mechanism for detecting these combinations in order to avoid unnecessary overhead.



US -----> CSS OPEN request

CSS -----> SS request for storage site

SS -----> CSS response to previous message

CSS -----) US response to initial message

Figure 4.2

4.2.1.4. Pathname Search

All pathnames start either from the root directory (/) or the current working directory of the process that presents the name. The search procedure consists of:

1) obtaining the <LFGN,FDN> of the starting directory from the appropriate inode.
2) an internal open to the directory (following the open protocol of example-1)
3) protection checks

4) if the pathname is not completely parsed the inode number of that component is read from the directory to continue the path search (the initial directory is closed).

If the directory search involves remote sites, reading the directories is done by using a read protocol that is controlled by the CSS.

4.2.2. Discussion of the Identifying Mechanism

4.2.2.1. Structure of Identifiers

There are two kinds of identifier structures in LOCUS. The user views a hierarchical name space that is used to logically organize his files. At the low level these identifiers are mapped to identifiers with an absolute structure ((LFGN,FDN)). After the mapping is done the operating system continues to utilize the absolute structure identifier, avoiding repetitive directory searches.

4.2.2.2. Homogeneous Name Space

A homogeneous name space is inherently provided in LOCUS because all machines run the same software. But even with a different configuration the identifying mechanism would provide for a homogeneous name space. The operating system is responsible for searching through directories that

compose the path name. The user perceives no difference between a remote and local object.

4.2.2.3. Passing Identifiers

Since the operating system binds pathnames to their respective low level identifiers as soon as possible, all further references to the object are made using its internal identifier. The internal identifier can easily be passed among processes. A uniform naming convention is used among all hosts.

4.2.2.4. Mapping

One-to-one mapping is supported in LOCUS, i.e., a path-name will be mapped uniquely to one logical file. It should be recalled that several copies of a file are associated with one <LFGN,FDN>, but this is supported by other mechanisms of LOCUS.

4.2.2.5. Location Transparency

Location transparency is provided by both levels of identifiers. The internal identifiers of LOCUS are globally unique and are location independent. As mentioned previously the components of a pathname are not related with physical locations of the directories. There is no need to refer to a specific site when requesting services and the user is not aware when a remote access is made.

4.2.2.6. Evolution

The identification mechanism and directory structure of LOCUS allow smooth evolution of the system. Pathnames have inherent characteristics of hierarchical structures, thus, in order to add a new host directories would have to be updated.

4.3. R* SYSTEM

4.3.1. Background and Major Features

The R* system is a distributed relational database developed at the IBM San Jose Research Laboratory in California. The star (*) in R* (pronounced R star) comes from the Kleene Star operator denoting an arbitrary number of R's. Queries to the databases are formulated using a high level language called Structured Query Language (SQL).

The naming mechanism used to access data objects (e.g. relations, fields) will be emphasized here, other aspects of R* are discussed in [Ceri84, Dani82, Date83].

The main objective of the R* system is to provide a distributed database composed of cooperating but autonomous sites. Site autonomy is achieved when a site has control over its own data, even when it cannot communicate with other sites in the network.

Site autonomy requires that the system be able to evolve and continue operations in the event of the addition (or deletions) of sites. It also should be possible for an existing site to access data objects from another site in a simple form. In the R* system, for two sites to be able to share data objects, the database administrator (person responsible for administrative updates) at each site must update their respective database (only once). Information

such as the network address and authentication parameters is included.

Site autonomy has several characteristics that are similar to those desired in an identification mechanism (easy evolution, location transparency, ability to retain existing software etc.), thus it is expected that identifiers in R* have these characteristics.

4.3.1.1. Objectives of the Naming Scheme

The naming scheme of R* has basically six objectives:
1) the same identifier can be used by different users to reference different data objects,

2) different identifiers can be used by different users to reference the same data object,

3) the same identifier, in the same program or query, may be resolved differently on different occasions (context switching),

4) global(system wide) name resolution information will not be required at any single site, i.e., names are to be resolved in a distributed fashion,

5) location independence,

6) identifiers can be location dependent, i.e, a specific

data object at a specific location can be accessed.

It can be noticed that the goals of the identification mechanism of R* are similar to and in several cases identical to, the goals discussed in chapter 1. R* meets the above objectives by implementing an identifying mechanism that has two types of identifiers: System Wide Names (SWN) and Print Names (PN).

4.3.1.2. Names

SWN's uniquely identify data objects in the system. A SWN has four components:

USER@USER_SITE.OBJECT_NAME@OBJECT_SITE

OBJECT_SITE is the network name of the database on which the data object was created. The OBJECT_SITE is the recipient of a descriptor of the data object (if the object exists). This component of the name is absolute, and a centralized authority must ensure that no two installations use the same site name for their databases.

The OBJECT_NAME part of the SWN is selected by the user that creates the name and should be representative of the function or purpose of the object. A data object is usually referenced by its OBJECT_NAME.

The USER and USER_SITE parts of the SWN distinguish different data objects created by different users that use the same OBJECT_NAME, which accomplishes objective #1 of the naming mechanism.

A print name (PN) is a character string chosen by the user to identify a data object. A name resolution facility maps the user's PN into a SWN. PN's have the same structure as the SWN but the only required component of a PN is the ⟨OBJECT_NAME⟩.

4.3.1.2.1. Print Name Resolution

The PN resolution mechanism in R* includes what are called synonyms. Each site maintains a synonym catalog for each user registered at the site. The synonym catalog maps simple OBJECT_NAMES and PNs to their SWN. Thus, synonym catalogs allow different users to use different PNs to reference the same data object(objective #2). Each user is responsible for maintaining his synonym catalog. For example, a user creating an entry in his catalog would use the following SQL statement:

```
DEFINE SYNONYM students
            AS  ComputerScience@RIT.cstudents@OneLambDr
```

The resolution of a PN to its SWN is discussed in detail in

Appendix-B.

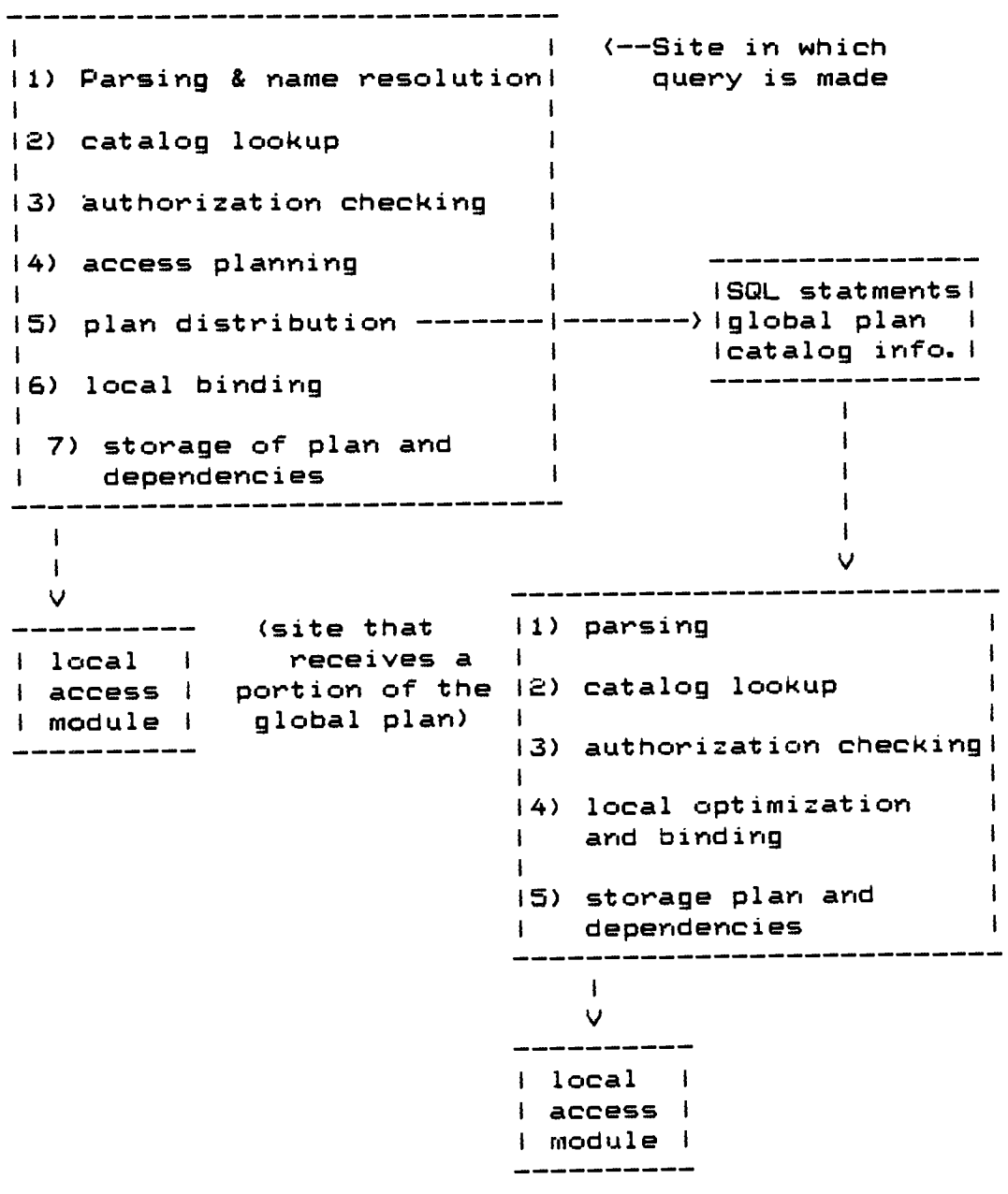
When all four components of the PN are given the user is making reference to a specific object at a specific location (objective #6), on the other hand, when some components are not present, the PN is resolved to different SWN's depending on the current context (default synonym catalogs are used, see steps 3a and 4a of figure B.1 in Appendix-B). For example, a programmer can have an account used for testing new programs (e.g. USERID=TEST-ACCT). A PN such as TABLE1 would be resolved using the synonym catalog named TEST-ACCT. Later when the program is to be used in regular production, the same name (TABLE1) can be resolved using the production synonym catalog.

4.3.1.3. Catalogs

In order to understand the structure and use of catalogs in R* it is necessary to observe how queries are resolved in the system.

Queries can be either compiled and stored for repeated executions or dynamically compiled for ad hoc queries. When a query is submitted at one site, several steps are followed to produce a local access module (see figure 4.3).

Name resolution in step 1 refers to the mapping of PNs to SWNs within the context of the user in the site



Query compilation as presented in [Ceri84].

FIGURE 4.3

originating the query. Catalog lookup is the process of actually locating the data object in the system. The database manager is responsible for finding this location and routing the request to the appropriate site. Authorization is checked locally in order to support site autonomy; each site involved is responsible for authorizing access to local data objects. After referenced data objects are located, the query optimizer selects the most efficient way to perform the query (step #4). The query plan is distributed to the site in which it is to be executed. Local bindings of SWNs to low level objects and their access paths (e.g. indexes) are then made (step #6). Finally the plan and the local data objects used (dependencies) are stored at the corresponding site.

4.3.1.3.1. Catalog Structures

Catalogs in R* are distributed without overlap, however each site maintains a cache of information about entries used from remote sites. Thus, each site stores entries for local objects only. A catalog entry of a data object contains all the information about the object needed for the construction of the low level plan, which is:

1) data object's type (relation, field, etc.) and its schema (number and type of columns)

2) where and how the data object is stored

3) the low level name (addresses) of the data objects implementation

4) the access path to reach the data object

5) statistics of the data object

The OBJECT_SITE component of a SWN identifies the site in which the catalog entry of the object resides. If an object migrates to another site, the entry in the OBJECT_SITE will contain the name of the site(s) in which the object currently resides. Thus, the OBJECT_SITE component will remain unchanged until the object is deleted. An access to an object will require at most two remote accesses (however, a site can cache the remote entry for future accesses). When a site requests a remote object either the entire catalog entry or site address (when the object has been relocated) is returned.

Each site maintains a catalog of dependencies. In this catalog the binding of data object names to low level objects (e.g. implementation of the object) and the access path needed to reach them are associated with the compiled queries. At execution time, locally stored query fragments are executed to produce the query results.

4.3.1.3.2. Distributed Data Objects

The R* catalog structure supports three types of distribution mechanisms: horizontal partitioning, vertical partitioning and replication of objects.

When horizontal partioning is used, a relation is divided based upon the value(s) of selected field(s), and stored at different sites. For example,

```
DISTRIBUTE TABLE students HORIZONTALLY INTO
    city-stud WHERE location = 'Downtown'
                IN SEGMENT stud-seg@Roch.DownTown
    Rit-campus WHERE location = 'Henrietta'
                IN SEGMENT stud-seg@Henrietta
```

would distribute the records contained in "students" to two different sites.

Vertical partitioning stores different fields (projections) of a relation at different sites. The system maintains and supplies a correlation field in order to join the fields of a record. This type of partitioning is efficient when different sites frequently use a specific subset of fields of the entire record.

Replication is used to improve locality, thus improving response time and availability of data. For example, the SQL statment:

```
DISTRIBUTE TABLE students REPLICATED INTO
  city-copy IN SEGMENT stud-seg@Roch.DownTown
  campus-copy IN SEGMENT stud-seg@Henreitta
```

will produce two copies of the object "students" and store them at two different sites.

4.3.1.4. Catalog Entries and Processing of Distributed Objects

When a data object is distributed, its structure has the form of a tree in which the leaves correspond to the fragments of the object. Figure 4.4 illustrates a distributed relation.

The birth site (OBJECT_SITE) of the data object contains the whole distribution tree in order to locate the fragments when a request is made. Thus, when a site accesses a distributed data object it directs its request to the OBJECT_SITE normally. The birth site returns the entire tree of catalog entries (in the case of fragments that do not reside in the birth site, their location is returned). The requesting site would then request the fragments from the sites in which they actually reside.

When the distribution of a data object changes, the version number associated with all catalog entries is incre-

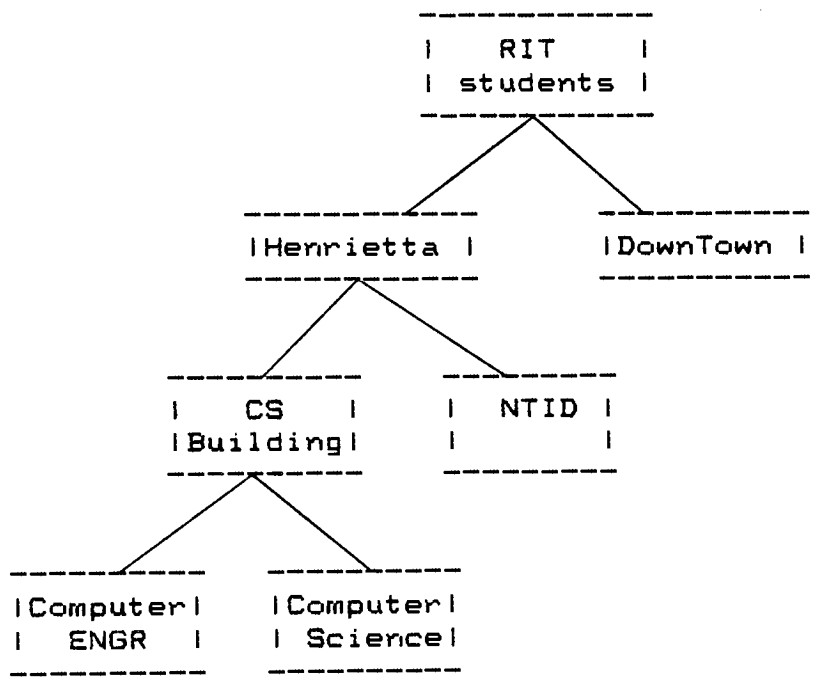


FIGURE 4.4

mented. If the access path of a stored fragment is altered, then the version number of that fragment's entry is incremented.

4.3.2. Discussion of the Identifying Mechanism

4.3.2.1. Structure of Identifiers

Identifiers in R* have an absolute structure. A SWN unambiguously identifies a single object in the system. PN

provide a "fiendly" way for the user to create and use names meaningful to him.

4.3.2.2. Homogeneous Name Space

The R* naming scheme provides a homogeneous name space. PNs are mapped into SWNs, thus the advantages of the Mapping Method are obtained.

4.3.2.3. Passing Identifiers

Passing identifiers in R* may not normally be done, but based on the absolute structure of SWNs, it would be easy to do so. Naming conventions are the same for all sites, i.e, the resolution of PNs is performed by the same procedure on all sites.

4.3.2.4. Mapping

By the use of synonym catalogs mappings of the type many-to-one and one-to-many (ids-to-data object) are provided.

4.3.2.5. Location Transparency

Location transparency is one of the explicit goals of R*. As mentioned previously, the user is not aware of whether an access is made to a local or remote object. The resolution of PNs make boundaries of different sites transparent to the user. The distributed database system supports relocation of objects by updating system catalogs to

indicate the location to which the object was moved. The user may reference an object using its SWN, thus location dependent identifiers are also supported.

4.3.2.6. Protection

The authorization mechanism (protection) is done at each site independently. Identifiers do not provide protection information.

4.3.2.7. Evolution

As long as the interfaces between the database managers follow a specific communication protocol they can be connected. This characteristic eases the addition of new databases. The naming mechanism of R* also allows the addition of new sites. The structure of SWNs (even though they are absolute) has some properties of a hierarchical structured identifier, thus distributed creation of identifiers is supported. Since PNs are mapped to SWNs using the basic concept of the Mapping Method they also support the evolution of the system.

4.3.2.8. Other Remarks

R* is an experimental project, therefore many problems must be resolved. The naming mechanism, however, is excellent (within the distributed database environment) because it has most of the characteristics that an identification mechanism should have in order to improve performance and

allow a distributed system to meet the goals discussed in chapter 1.

4.4. ROSCOE

4.4.1. Background and Major Features

ROSCOE is an operating system implemented at the University of Wisconsin that supports a network of microcomputers. All the processors are identical and run the same operating system kernel. The communication between processors is done by message passing.

A major goal of the system is to appear to the end-user as a single powerful machine. Processes are able to communicate with a remote process in the same way as with a local process.

The design of ROSCOE provides for many operating system functions to be implemented by processes instead of the kernel. For example, each machine in the network has a file manager process and a resource manager process which execute their functions on a request/reply message scheme. The kernel provides various services for user programs that are requested by "service calls". The fundamental components of ROSCOE are links, processes and messages.

4.4.1.1. Links

The link concept is vital to ROSCOE. Links are a one-way logical connection between two processes, the "holder"

and the "owner" (as illustrated below in figure 4.5).

The owner process has the capability of creating a link. When a link is created, the owner specifies a channel on which incoming messages will be received (channels can be seen as port names). Thus a link can be partitioned into "sub-links". In order for a holder process to be able to communicate and request service over a link, it must posses the channel number and the link number that identifies the link. The holder process must have previously obtain these over a pre-existing link.

The receiving process (owner) can control the channels on which it will accept incoming messages at any given point in time. The owner process is blocked until a message arrives on one of the specific channels. In addition, the owner process can specify certain restrictions to the holder

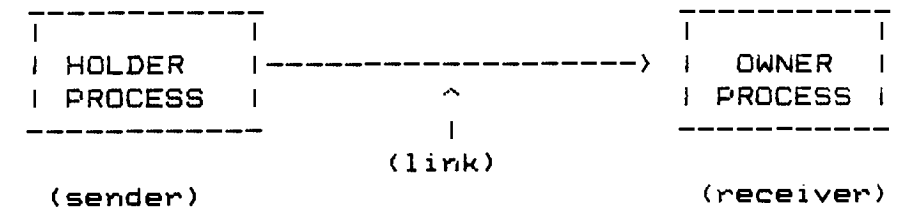


FIGURE 4.5

processes, such as preventing the copying of the link number and allowing the use of the link only once.

The kernel provides the necessary context for the resolution of the link number, which uniquely determines the link. Each holder process is allocated a table which contains all the links currently sustained by that holder. The link number is an index to this table. When messages are forwarded to the owner process the kernel will tag the message with the channel over which it was sent. Additional information (such as restrictions) is also stored in the kernel supported tables.

4.4.1.2. Processes

Processes in ROSCOE are communicating entities that execute specified functions. For example, a terminal driver process is present at every processor. All input/output to terminals is done through the terminal driver by passing messages.

The resource manager process is the "umbilical cord" that connects a newly created process to its local system. All the resource managers are connected by a network of links. Thus a process can migrate to a remote processor when resources are not available on the processor on which it was

created**. The user is not aware of this.

File management is done by a file manager process; there is one on each machine. A tree-structured directory at each processor is used by the file manager process to locate files. Processes that wish to open a file must send a request to the file manager. The file manager then creates a link and returns a reply message with the link number. Thus, an open file is represented by the established link. The file is closed when the link is destroyed. The users only see a process that replys to read/write requests that are sent over a link.

In order to remove from the user the burden of complicated protocols, library routines are implemented. The functions available provide a simple way to use utility processes and ease the writing of programs. For example, message passing is provided by a function named "call". The "call" function will:

1) create a reply link
2) send the message to its destination
3) wait for the reply (block the process)
4) decode the returned message.

Additional primitive functions that interface with the file

**The ROSCOE system allows process migration but it is restricted to the same address location in all processors, i.e., memory relocation is not supported.

process manager are also available (e.g. requesting directory information, open/close/create/delete files, etc.).

4.4.1.3. Messages

Specific system routines are available to reformat input/output data to fit within message passing conventions. For example, for terminal I/O a routine will accept input strings and divide them into fixed size messages. Another routine accepts the messages and assembles them into a buffer.

A specific kind of message is referred as a notification. This type of message is created and sent by the kernel to inform the owner process (receiving part of a link) of a change in the status of the link. As an example, an owner process can specify that it should be notified when its link is copied. Another situation is an owner process specifying that it is to receive a notification each time one of its links is destroyed.

Messages are queued by the kernel in a pool of buffers that is shared among all local processes.

4.4.2. Discussion of the Identifying Mechanism

4.4.2.1. Structure of Identifiers

The identifiers in ROSCOE are the link and channel numbers. These have absolute structures, thus unambiguously identifying each communication path and sub-channel respectively. Link numbers are positive integers.

4.4.2.2. Homogeneous Name Space

The architecture of the ROSCOE system provides a homogeneous name space. In the same way the evolution of the system is not obstructed by the naming mechanism.

4.4.2.3. Passing Identifiers

Passing identifiers from one process to another is easily done by including the link number in a message that is sent. The sender of the message must already have possession of the link number. The kernel will proceed to update the table (context) of the new holder process; the recipient of the link becomes the holder of the link. Notice that if the original process retains a copy of the link number, both holder processes would be connected to the owner process.

4.4.2.4. Mapping

The naming of links allows a mapping of many-to-one (name-to-objects) and one-to-one. As mentioned above, by

passing the link number, various processes can be connected to one process (as illustrated below in figure 4.6).

4.4.2.5. Location Transparency

Location transparency is achieved by the use of link numbers. A holder process does not know the address or name of the server process, and cannot distinguish whether it resides locally or on a remote host. In addition to location transparency, the use of link numbers provides several advantages. Generic services can be implemented because the holder process does not need to know which process is replying to his messages. Another advantage is the level of

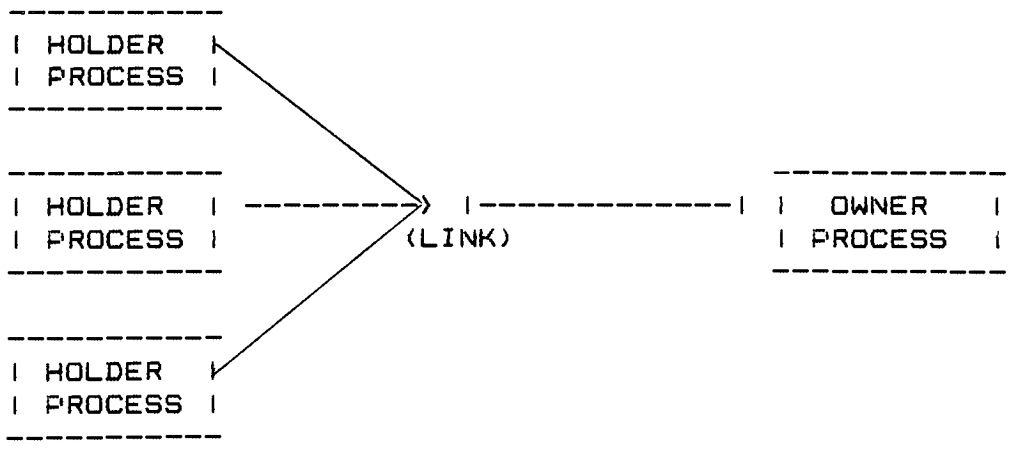


FIGURE 4.6

abstraction that is obtained between the service provided and the actual implementation of the service. For example, when the file manager process receives a request to open a file, instead of creating a new process to handle the incoming request it creates a new link pointing to itself. The holder process is unaware of this procedure. If future changes are made to the implementation process the user program does not need to be changed.

4.4.2.6. Protection

The naming mechanism supports protection in the system. Several types of links exist that enhance protection. A "reply link" is allowed to be used only once, thus it restricts the holder to sending exactly one message. An owner process can request notifications in the event of a holder copying, passing or destroying its link number. Notifications allow owner processes keep records of the number of processes that have a copy of its link number. Restricting access to files is accomplished by restricting the distribution of the link number of the file to authorized processes only.

4.4.2.7. Other Remarks

The original implementation of ROSCOE has several aspects that are subjects of further research. Migration of processes is impeded in one way due to the naming scheme. Even though a holder refers to the owner process by link

number rather than the name of the process, the kernel binds link numbers to their corresponding processes using the process name (utilizing the holder's link table as context). Thus when a process is moved to another host an updating mechanism is needed to correct the existing bindings. An approach to this situation is to provide a forwarding address in order to find the new location of the process. In this approach the chain of forwarding addresses might get too long and therefore adversely affect the performance of the system.

ROSCOE can be considered as a pioneer system because it was originally implemented in the mid 70's, thus it provided a "test bed" for different approaches used in distributed systems.

4.5. Concluding Remarks

The identification mechanisms discussed are designed for their respective environments, however, many characteristics are common to all. The basic concepts of the generalized identification methods and identifier structures discussed in chapter 2 are used in these identification mechanisms.

The high level identification mechanism of LOCUS has the basic characteristic of the hierarchical concatenation

method. R*'s high level identification mechanism is based on the mapping method. ROSCOE and TRIX are more related to the dynamic allocation method.

The systems discussed are implemented on networks of hosts that run homogeneous system software. This fact facilitates the evolution of the systems and inherently helps to provide an homogeneous name space. Nevertheless, the identification mechanisms have the characteristics needed to be used in an heterogeneous environment.

Table 4.1 summarizes the main characteristics of the systems. As can be seen, identification mechanisms can use more than one structure, thus obtaining many advantages from each one.

Common to all the identification mechanisms is the use of absolute structured identifiers. The uniqueness of these identifiers is obtained in different forms. In TRIX and ROSCOE it is the responsibility of the system to generate a unique identifier when an "incarnation" of an object is needed. In the R* system the uniqueness of the SWN (System Wide Name) is obtained by the concatenation of several fields. LOCUS concatenates two fields. Absolute structured identifiers are helpful when passing identifiers because the naming resolution is common to all the hosts on the system.

The mapping of identifiers to objects varies according to the context used to resolve names, the structure of

identifiers and the configuration of the system.

Identifiers in TRIX and ROSCOE influence the protection mechanism of their systems by limiting the distribution of identifiers and by assigning types to identifiers respectively.

To state that one identifying mechanism is superior to another would be wrong. Each of these mechanisms has their own goals and serves different purposes.. The techniques in each system can be seen as bases for further development of better identification mechanisms.

Table 4.1 (part 1)

Characteristics	Naming Mechanisms used in TRIX	LOCUS
Structure of Identifiers	. absolute . hierarchical	. absolute . hierarchical
Homogeneous Name Space	. homogeneous software . absolute identifiers	. homogeneous software . pathnames
Passing Identifiers	. all hosts have the same naming convention . stream descriptor can easily be passed	. (LFGN,FDN) can easily be passed
Mapping (id-to-obj)	. many-to-one . one-to-one	. one-to-one
Location Transparency	. supported	. supported
Protection	. processes control access to objects by limiting the distribution of stream descriptor	. not supported
Evolution	. pathnames	. pathnames and directory structure support evolution

Table 4.1 (part 2)

Characteristics	Naming Mechanisms used in R*	ROSCOE
Structure of Identifiers	. absolute	. absolute
Homogeneous Name Space	. supported by . the mapping of PN to SWN	. homogeneous software
Passing Identifiers	. not usual but can be done	. link numbers . can be passed
Mapping	. many-to-one . one-to-many	. many-to-one . one-to-one
Location Transparency	. supported*	. supported
Protection	. not supported	. directly supported by typing of links. . restricting link distribution
Evolution	. supported based on mapping method	. supported

* Location Dependency is also supported

5. Conclusions

Research and implementation of distributed systems have increased over the past years. Decreases in the cost of hardware and system software development have made feasible the use of physically disperse processors connected by a network. A global resource naming mechanism is needed because of the expansion of services.

The selection of an appropriate identification mechanism can help the the system achieve the following goals:

* Sharing objects
* Relocation of objects without changing references
* Provision of generic services
* Protection of objects
* Error control and synchronization
* Evolution of distributed systems by the integration of existing applications and systems
* The capability of easily passing identifiers among objects, thus allowing objects refer to other objects by their identifiers
* A logical single machine in which all resources are homogeneously identified (logical integration of physically disperse resources).

An identification mechanism should support at least two levels of identifiers. A user-oriented level which will ease

the reference of resources by a human, and a machine oriented level that can easily be manipulated and stored by processors.

Overall performance of the system will be affected by the identification mechanism. The use of additional levels of identifiers provides the indirection needed to achieve higher availability of resources, however the price paid for the availability is the overhead introduced by the dynamic binding.

The identification mechanism used will also have an effect on the amount of memory resources needed to store the context used in the resolution of identifiers.

Reliability and robustness of the system can be enhanced by the identifying scheme. By providing location transparency, resources can be relocated to different processors in order to continue offering services in the event of a processor failure.

This dissertation has presented the advantages and drawbacks of various identification schemes, different alternatives used in naming, and tradeoffs between design alternatives. The potentials of naming have been discussed with the purpose of demonstrating the influence of the naming mechanism on the overall performance and development of distributed systems.

6. APPENDIX-A

Properties in the Clearinghouse

Each name in the Clearinghouse is mapped into a set of properties. A property consists in an ordered tuple of the form:

{(Property name, property type, property value)}.

The property name unambiguously identifies the property itself. Property type can either be equal to 1 (meaning a group of objects) or equal to 0 (meaning an individual object). If the property type is equal to 1 then the property value is treated as set of names (of the format LOCAL@DOMAIN@ORGANIZATION). If it is equal to 0 then the property value is treated as an uninterpreted block of data. A name can have an arbitrary number of properties attached to it.

EXAMPLES

1) Mapping a name to an address:

Printer-A@Business@RIT -->

{("printer", 0, network address of
the printer named
Printer-A)}

2) Mapping a name to a set of printer (group)

```
Printers@ComputerScience@RIT--> {("list of printers",1,
                                  {"LPR@CV@RIT", "RPR@CV@RIT",
                                   "LPR@VP@RIT", "RPR@VP@RIT"})}
```

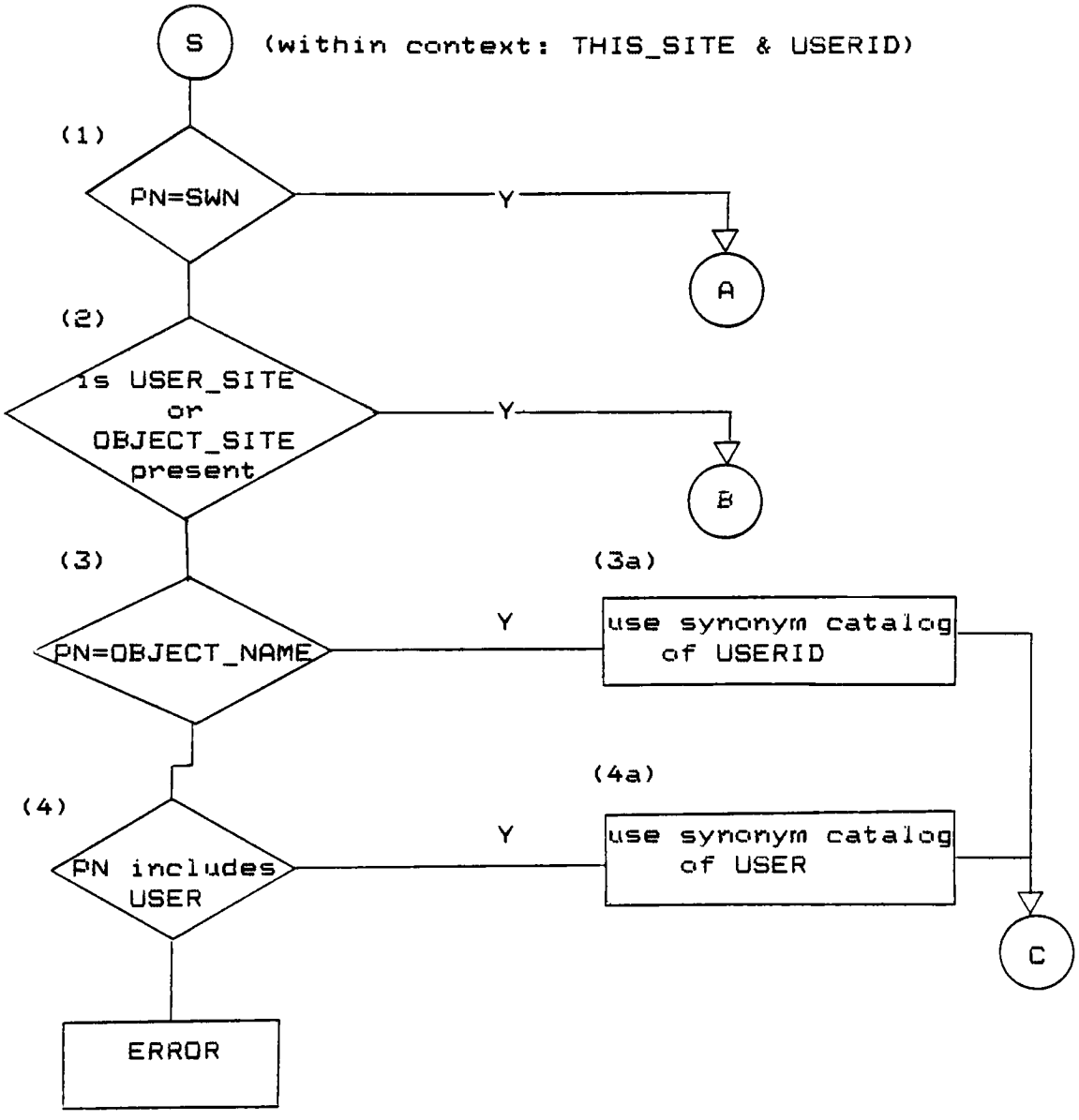
3) A user with more than one property associated.

```
Wilfredo@ComputerScience@RIT -->
                              {("USER",0,Wilfredo Rodriguez),
                               ("Password",0,XXXX),
                               ("Mail Account,0,WXR3245),
                               ("PrinterNames,1,{LPR@CV@RIT,
                                                 RPR@CV@RIT})}
```

Individual objects such as a printer are mapped into addresses (example 1) while groups are mapped into sets of names (example 2). This scheme of properties allows an object to be mapped not only to its network address but also to include more information related to the object. For instance, in example 1 information related to the printer's location, size of the forms used etc. can be incluced.

7. APPENDIX-B

Print Name Resolution Flow Chart



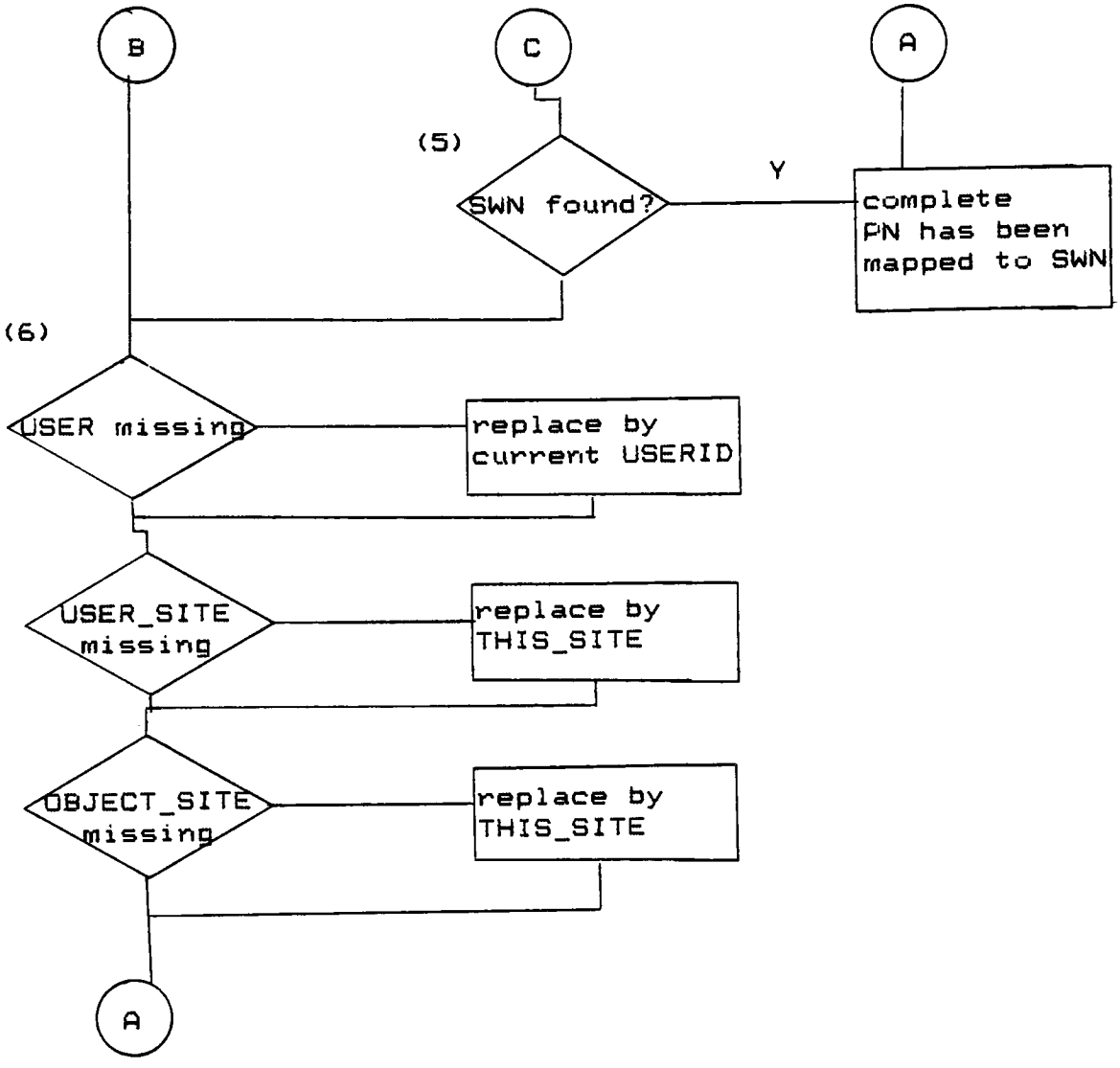


FIGURE B.1

Print Name Resolution Flow Chart

As seen from figure B.1

1) if the PN has all four components no further resolution is done

2) if USER_SITE or OBJECT_SITE is present the local synonym catalogs are not used. Missing fields are replaced by default values(6).

3) synonym mapping

3a- if the OBJECT_NAME component is the only present, the default catalog used is of USERID (taken from login procedure).

4) if the PN includes USER component, then the USER synonym catalog is used to map the PN to SWN

5) if the SWN is not found (i.e. not match), default values are used for missing components(6)

After the PN is resolved to its SWN, access to the data object's descriptor can be made.

8. APPENDIX-C (Glossary)

access path : an access path is a connection between objects. The connection can physical or logical. For example, the route that a message must travel within a subnet is an access path.

bind: binding is the process of mapping an identifier to an object by means of a context.

context: is a set of binding of identifiers to objects, that is, a partial mapping from some names to some objects of the system. A file catalog is an example of a context.

host: a host is an autonomous computer system capable of processing data and executing programs by itself.

inode: an inode is file descriptor that contains information such as: the owner's user and group identification, protection bits, physical addresses of the contents of the file, and file size.

large address space : a large address space in this chapter is one addressed by 64 bits or more.

local operating system: a local operating system is one that resides locally in a computer system and manages the resources of that system.

mapping function: a mapping function is the method used to associate an identifier with an object.

modules : a module is a self-contained object that can contain data and is capable of processing it. [Feld79]presents a high level programming language designed for a distributed computing environment. Programs are composed of modules that communicate by a message passing system.

object: an object is a system component that is considered to be worthy of a distinct identifier.

abstract objects: abstract objects are software structures such as: processes, databases, files, directories etc.

real object: real objects are hardware devices such as: processors, secondary storage, I/O devices etc.

process: a process is an active object (in state of execution).

relational data base: is a database in which the conceptual files are all relations; a relation is defined as a file in which all records are unique and have the same number and types of fields, i.e., records have fixed length and format.

resolve: the process of locating an object by its name. This is accomplished by using a specific mapping function and context.

9. BIBLIOGRAPHY

[Abra80] Abraham, S.M., Dalal, Y.K., "Techniques for Decentralized Management of Distributed Systems," Compcon 80, (Spring 1980), 430-437.

[Andr82] Andrews, Gregory R. "Distributed Programming Languages," Proceedings of the ACM 82 Conference, (October, 1982), 113-117.

[Bish79] Bishop, Matt and Snyder, Lawrence. "The Transfer of Information and Authority in a Protection System," Communications of the ACM, (1979), 45-54.

[Brad83] Bradley, James. Introduction to Data Base Management in Business. New York: CBS College Publishing, 1983.

[Brow84] Browne, James C. et al. "Zeus" An Object-Oriented Distributed Operating System for Reliable Applications," Proceedings of the 1984 Annual Conference of the ACM, (October, 1984), 179-188.

[Ceri84] Ceri, Stefano and Pelagatti, Giuseppe. Distributed Databases Principles and Systems. New York: McGraw-Hill,

Inc., 1984.

[Cham80] Champine, George A. Distributed Computer Systems Impact on Management, Design, and Analysis. Amsterdam: North Holland Publishing Co., 1980.

[Chri83] Christian, Kaare. The UNIX Operating System. New York: John Wiley & Sons, 1983.

[Craf83] Craft, Daniel H. "Resource Management In A Decentralized System," ACM Operating System Review, XVII (October 1983).

[Dani82] Daniels, Dean et al. "An Introduction to Distributed Query Compilation in R*," Research Report RJ3497 (41354), IBM Research Lab., San Jose, CA, (June, 1982), 1-21.

[Date83] Date, C. J. An Introduction to Database Systems. Reading: Addison-Wesley Publisning Co., 1983.

[Donn79] Donnelly J.E.. "Components of a Network Operating System," Computer Networks, III (No. 6), (December, 1979) 389-400.

[Deit84] Deitel, Harvey M. An Introduction to Operating Systems. Reading: Addison-Wesley Publishing C., 1984.

[Ens178] Enslow, P. H. Jr. "What is a "Distributed" Data Processing System?," Distributed Computing: Concepts and Implementations. Paul L. McEntire: IEEE Press, 1978.

[Fors81] Forsdick, Harry C. et al. "Operating Systems for Computer Networks," in Liebowitz, Burt and Carson, John (Eds.) 1981.

[Fors78] Forsdick, Harry C. et al. "Operating Systems for Computer Networks," Computer, XI (January, 1978), 48-57.

[Feld79] Feldman, Jerome A. "High Level Programming for Distributed Computing," Communications of the ACM, XXII (June, 1979), 353-368.

[Feld78] Feldman, J. A. et al. "Programming Distributed Systems," Computer Science Engineering Research Review 1977-1978.

[Gray82] C Gray Girling, "Object Representation on a Heterogeneous Network," ACM Operating Systems Review, XVI

(October, 82), 49.

[Gert83] Gertner, Ilya and Lindenberg, Robert. "Initializing Replicated Name Servers in a Wide Area Network," IEEE Proceedings of the Third Symposium on Reliability in Distributed Software and Database Systems, (October, 1983), 90-94.

[Katz79] Katzan, Harry Jr. Distributed Information Systems. New York: Petrocelli Book, 1979.

[Land83] Landweber, L. et al. "Architecture of the CSNET Name Server," Communication of the ACM, (March 1983), 146-153.

[Lau84] Lau, Francis C.M. "Two-Part Names and Process Termination." Operating System Review SIGOPS/ACM, (July 1984), 28-30.

[Leac82] Leach, Paul J. et al. "UIDS as Internal Names in a Distributed File System," ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, (August 1982), 34-41.

[Lind80] Lindsay, Bruce. "Object Naming and Catalog Management for a Distributed Database Manager," IBM Research

Report RJ2914, (August 1980).

[Mart81] Martin, James. Computer Networks and Distributed Processing Software, Techniques and Architecture. Engle Cliffs N.J.: Prentice-Hall Inc., 1981.

[McGl78] McGlynn, Daniel R. Distributed Processing and Data Communication. New York: Wiley-Interscience Publication, 1978.

[McQu78] McQuillan, J.M., "Enhanced Message Addressing Capabilities For Computer Networks," Proceedings IEEE, LXVI (November 1978), 1346-1370

[Oppe81] Oppen D.C. and Dalal Y. "The Clearinghouse: A Decentralized Agent for Locating Named Objects in Distributed Environment," Office Products Division Technical Report OPD-T8103, XEROX (October 1981).

[Ness82] Nessett D.M."Identifier Protection in a Distributed Operating System," ACM Operating Systems Review, (Jan 82) XVI, 26.

[Pash82] Pashtan, Ariel. "Object Oriented Operating Systems:

An Emerging Design Methodology," Proceedings of the ACM 82 Conference, (October, 1982), 126-131.

[Pope81] Popek, G. et al."LOCUS: A Network Transparent, High Reliability Distributed System," Proceedings of the Eight Symposium on Operating Systems Principles, (December, 1981), 137-147.

[Pouz82] Pouzin, L. et al. The CYCLADES Computer Network. Amsterdam: North-Holland Publishing Co., 1982.

[Rama83] Ramamritham, Krithivasan et al. "Primitives for Accessing Protected Objects," IEEE 3rd Symposium on Reliability in Distributed Software and Database Systems, (October, 1983), 114-122.

[Reed78] Reed, D.P. "Naming and Synchronization in Decentralized Computer Systems," Technical Report MIT/LCS/TR205, (September, 1978).

[Salt78] Saltzer J.H., "Naming and Binding of Objects," Lecture Notes in Computer Science, New York: Springer-Verlag, Chapter 3 99-208 (1978).

[Shoc78] Shoch, John F. "Inter-Network Naming, Addressing, and Routing," Proceedings 17th IEEE Computer Society International Conference (COMPCON), (September, 1978), 72-79.

[Solo79] Solomon, Marvin H. and Finkel Raphael A. "The ROSCOE Distributed Operating System," Proceedings of the 7th Symposium on Operating Systems Principles, (December, 1979), 108-114.

[Solo78] Solomon, Marvin H. and Finkel Raphael A. "ROSCOE: A Multi-Microcomputer Operating System," Computer Science Technical Report No. 321 C.S.D. University of Wisconsin Madison, (May 1978).

[Tane81] Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs N.J.: Prentice-Hall Inc., 1981.

[Tane81a] Tanebaum, Andrew S. "Network Protocols," Computing Surveys. XVIII (December, 1981), 453-489.

[Terr84] Terry, Douglas B. "The COSIE Name Server," IBM Research Report RJ4161 (45949), Jan. 1984.

[Terr83] Terry, Douglas B. and Andler, Sten "The COSIE

Communication Subsystem : Support for Distributed Office Applications," Research Report RJ4006 (45054), August, 1983.

[Thom73] Thomas R. H. "A Resource Sharing Executive for the ARPANET," AFIPS Conference Proceedings, (1973), SJCC 155-163.

[Trip83] Tripathi, Anand et al. "An Object-Oriented Design Model for Reliable Distributed Systems," IEEE Proceedings of the 3rd Symposium on Reliability in Distributed Software and Database Systems, (October 1983), 90-94.

[Walk83] Walker, Bruce et al. "LOCUS Distributed Operating System," Proceedings of the 9th Symposium on Operating Systems Principles, XVII (October, 1983), 49-70.

[Ward80] Ward, Stephen A. "TRIX: A Network Operating System," COMPCON 80, Spring, (Feb.,1980), 344-349.

[Wats81] Watson, Richard "Identifiers(Naming) in Distributed Systems," in Lampson, W. et al. Lecture Notes in Computer Science: Distributed Systems-Architecture and Implementation. New York: Springer-Verlay (1981).

[Wilm83] Wilms, Paul F. et al. ""I wish I were over there": Distributed Execution Protocols for Data Definition in R*," ACM SIGMOD International Conference on Management of Data, XIII (May, 1983), 238-242.

[Zimm80] Zimmermann, H. "The ISO Model of Architecture For Open Systems Interconnection," IEEE Trans. Communication XXVIII (April, 1980), 212-219.

[Zimm78] Zimmermann, Pouzin H."A Tutorial on Protocols," Proceedings of the IEEE LXVI (Nov., 1978), 1346-1370.