The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment

DEREK C. OPPEN and YOGEN K. DALAL
Xerox Office Systems Division

The problem of naming and locating objects in a distributed environment is considered, and the *clearinghouse,* a decentralized agent for supporting the naming of these "network-visible" objects, is described. The objects "known" to the clearinghouse are of many types and include workstations, file servers, print servers, mail servers, clearinghouse servers, and human user. All objects known to the clearinghouse are named using the same convention, and the clearinghouse provides information about objects in a uniform fashion, regardless of their type. The clearinghouse also supports aliases.

The clearinghouse *binds* a name to a set of *properties* of various types. For instance, the name of a user may be associated with the location of his local workstation, mailbox, and nonlocation information such as password and comments.

The clearinghouse is decentralized and replicated. That is, instead of one *global* clearinghouse server, there are many *local* clearinghouse servers, each storing a copy of a portion of the global database. The totality of services supplied by these clearinghouse servers is called "the clearinghouse." Decentralization and replication increase efficiency, security, and reliability.

A request to the clearinghouse to bind a name to its set of properties may originate anywhere in the system and be directed to any clearinghouse server. A clearinghouse client need not be concerned with the question of which clearinghouse server actually contains the binding—the clearinghouse stub in the client in conjunction with distributed clearinghouse servers automatically finds the mapping if it exists. Updates to the various copies of a mapping may occur asynchronously and be interleaved with requests for bindings of names to properties; updates to the various copies are not treated as indivisible transactions. Any resulting inconsistency between the various copies is only transient: the clearinghouse automatically arbitrates between conflicting updates to restore consistency.

Categories and Subject Descriptors: C.2.3 [**Computer-Communication Networks**]: Network Operations—*network management*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed databases*; *network operating systems*; H.2.1 [**Database Management**]: Logical Design—*data models*; H.3.3 [**Information Storage and Retrieval**]:Information Search and Retrieval—*search process*; H.4.3 [**Information Systems Applications**]: Communications Applications—*electronic mail*

General Terms: Design

Additional Key Words and Phrases: Clearinghouse, names, locations, binding, network-visible objects, internetwork

1. INTRODUCTION

We introduce the subject matter of this paper by considering the role of the information operator, the "White Pages," and the "Yellow Pages" in the telephone system.

Authors' addresses: D. Oppen, 495 Arbor Road, Menlo Park, CA 94025; Y. Dalal, Metaphor Computer Systems, 2500 Garcia Avenue, Mountain View, CA 94043.

Consider how we telephone a friend. First we find the person's telephone number, and then we dial the number. The fact that we consider these to be "steps" rather than "problems" is eloquent testimony to the success of the telephone system. But how do the two steps compare? The second—making the connection once we have the telephone number—is certainly the more mechanical and more predictable, from the user's point of view, and the more automated, from the telephone system's point of view. The first step—finding someone's telephone number given his or her name—is less automatic, less straightforward, and less reliable. We have to use the telephone system's information system, which we call the *telephone clearinghouse.* If the person lives locally, we telephone "information" or look up the telephone number in the White Pages. If the person's telephone is nonlocal, we telephone "information" for the appropriate city. We always have to treat whatever information we get from the telephone clearinghouse with a certain amount of suspicion and as a "hint." We have to accept the possibility that we have been given an incorrect number, perhaps because the person we wish to call has just moved. We are conditioned to this and automatically begin calls with "Is this . . . ?" to validate the hint.

In other words, although making the connection once we have the correct telephone number offers few surprises, *finding* the telephone number may be a time-consuming and frustrating task. The electrical and mechanical aspects of the telephone system have become so sophisticated that we can easily telephone almost anywhere in the world. The telephone clearinghouse remains unpredictable and may require considerable interaction between us, as clients, and the information operator. As a result we all maintain our own personal database of telephone numbers (generally, a combination of memory, little black books, and pieces of scrap paper) and rely on the telephone system's database only when necessary.

The telephone clearinghouse provides another service: the Yellow Pages. The Yellow Pages maps generic names of services (such as "Automobile Dealers") into the names, addresses, and telephone numbers of providers of these services.

In brief, there are three ways for objects in the telephone system to be found: by name, by number, or by subject. The telephone system prefers to use numbers, but its clients prefer subscriber and subject names. The telephone clearinghouse provides a means for mapping between these various ways of referring to objects in the telephone world.

We move from the telephone system to distributed systems and, in particular, to interconnections of local networks of computers. Suppose that we want to send a file to our local printer or to someone else's workstation, or we want to mail a message to someone elsewhere in the internetwork. The two steps we have to take remain the same: finding out where the printer, workstation, or mail server is (that is, what its network address is), and then using this network address to access it. The internetwork knows how to use a network address to route a *packet* to the appropriate machine in the internetwork. So the second step—accessing an object once we know its network address—has well-known solutions. It is the first step—finding the address of a distributed object, given its name—that we consider here.

Why do we need names at all? Why not just refer to an object by its address? Why not just directly use the network address of our local file server, mail server, or printer? The reasons are much like those for using names in the telephone system or in a file system. The first is that locations are unappealingly unintuitive; we do not want to refer to our local printer by its network address 5#346#6745 any more than we want to refer to a colleague as 415-494-4763. The second is that distributed objects change locations much more frequently than they change names. We want a level of indirection between us and the object we wish to access, and that level of indirection is given by a name. (See also [1], [11], [12], [14], and [15].)

When a network object is referred to by name, the name must be *bound* to the address of the object. The binding technique used greatly influences the ability of the system to react to changes in the environment. If client software binds names to addresses *statically* (for instance, if software supporting printing has the addresses of the print servers stored in it), the software must be updated if the environment changes (e.g., if new print servers are added or old servers are moved or removed). If client software binds names to addresses *dynamically*, the system reacts much more gracefully to changes in the environment (they are not necessarily even noticed by the client).

The problems we address in this paper are, therefore, the related problems of how to name objects in a distributed computer environment; how to find objects, given their names; and how to find objects, given their generic names—in other words, how to create an environment similar to the telephone system's, with its notions of names, telephone numbers, White Pages, and Yellow Pages. Before leaving this introduction, we see how the telephone clearinghouse works.

1.1 Locating Telephone Subscribers

The database used by the telephone clearinghouse—the "telephone book"—is highly decentralized. The decentralization is based on physical locality: each telephone book covers a specific part of the country. It is up to the telephone clearinghouse client to know which telephone book is to be used.

This decentralization is partly motivated by size; there are just too many telephones for one common database. It is also motivated by the fact that people's names are ambiguous. Many people may share the same name, and corresponding to one name there may be many telephone numbers. Decentralizing the telephone clearinghouse is one way to provide additional information for disambiguating a reference to a person—there may be many John Smiths in the country, but we hope that not all are living in the same city as the John Smith whose telephone number we want. However, even by partitioning the database by city and by using other information such as street address, the telephone clearinghouse still may be confronted with a name for which it has several telephone numbers. When this happens, it becomes the client's responsibility to disambiguate the reference, perhaps by trying each telephone number until he finds the one he wants. The telephone clearinghouse cannot assume that names are unambiguous, and it leaves it to the client to resolve ambiguities.

1.2 Creating, Deleting, and Changing Telephone Numbers

Responsibility for *initiating* updates rests with the telephone users. However, the actual updating of the database is done by the telephone company. Users of the telephone clearinghouse have read-only access to the clearinghouse database.

Allocation of telephone numbers is the responsibility of the telephone company; the telephone company provides a *naming authority* to allocate telephone numbers.

The updating process deserves scrutiny because it helps determine the accuracy of the information given out by the telephone clearinghouse. The information is not necessarily "correct." Off-line "telephone books" are updated periodically and so do not contain recent updates. Even the on-line telephone directory used by information operators may give "old" information. One reason for this is that asking the operator for a telephone number and using it some time later to make a call are not treated by the telephone system as an indivisible operation: the directory may be updated between the two events. Another reason is that physically changing a telephone number and updating the database are asynchronous operations.

The partitioning of the telephone clearinghouse's database is not strict. The database is a replicated database. Copies of a directory may appear in different versions, and telephone directories for different cities may overlap in the telephone numbers they cover. Since the updating process is asynchronous, the database used by the telephone company may not be internally consistent.

The effect of this—information given out by the telephone clearinghouse does not necessarily reflect all existing updates—is that the information provided by the telephone clearinghouse can only be used as a *hint*. The user must accept the possibility that he is dialing a wrong number and *validate* the hint by checking in some way that he has reached the right person. However, the telephone company does provide some mechanisms for helping a user who is relying on an out-of-date directory, memory, or little black book. For instance, if a person moves to another city, his old telephone number is not reassigned to another user for some time, and during that period callers of his old number are either referred to his new number or are less informatively told that they have reached an out-of-service number.

1.3 Creating, Deleting, and Changing Subscriber Names

What we said above about updating telephone numbers generally applies as well to updating names, with one exception. The *choice* of name appearing in the telephone clearinghouse database rests with the holder of the telephone being named, and only the holder can request an update. (That is, you are permitted to choose under what name you will appear in the telephone directory, even if the name is ambiguous.) This raises an interesting issue: that of nicknames, abbreviations, and aliases. The above does not mean that we, as users of the telephone system, cannot choose our own name for you (a *nickname*), but only that the telephone company will not maintain the mapping of our name for you into your telephone number—it will only maintain the mapping of *your* name for yourself into your telephone number. We may have our own "little black book" containing our own relativized version of the telephone clearinghouse, but the telephone company does not try to maintain its accuracy. Similarly, the telephone clearinghouse does not necessarily respond to abbreviations of names. And finally, the telephone clearinghouse will handle aliases only if they are entered in its database. That is, the telephone clearinghouse allows names to be nonunique: a person may have more than one name.

1.4 Passing Subscriber Names and Telephone Numbers

Giving someone else a telephone number cannot raise problems because telephone numbers are unambiguous. (Of course, the telephone number may be incorrect by the time that person uses it.)

Giving a name to someone else is trickier since names are ambiguous. For instance, because the telephone clearinghouse database is decentralized, giving a name to an information operator in one part of the country may elicit a different response from giving it to one in another part of the country. In the telephone clearinghouse, names are context dependent. You can ensure that the person to whom you are giving a name will get exactly the same response only if you specify the appropriate telephone clearinghouse as well.

2. NAMING DISTRIBUTED OBJECTS

With this background, we return to the problem of designing a distributed system clearinghouse. A central question in designing such a clearinghouse is how to name the objects known to the clearinghouse.

2.1 Naming Conventions

A *naming convention* describes how *clients* of the naming convention refer to the *objects* named using the convention. The set of clients may overlap with the set of named objects; for instance, people are both clients of, and objects named, using the common first-name-middle-name-surname naming convention.

Our basic model for describing naming conventions is a directed graph with *nodes* and *edges*. Nodes and edges may be labeled. If node u has an edge labeled i leading from it, then $u[i]$ denotes the node at the end of the edge. (Edges leading from any node must be unambiguously labeled.)

We assume that each named object and each client is represented by exactly one node in the graph. With these assumptions, we need not distinguish in the rest of this section between nodes in the name graph, named objects, and clients of the naming system, and our problem becomes: What is the name of one node (a named object) relative to another (a client)? There are two fundamental naming conventions, each of which we now describe.

2.2 Absolute Naming

Under the absolute naming convention, the graph has only unlabeled nodes. There is a distinguished node called the *directory* or *root node*. There is exactly one edge from the directory node to each other node in the graph; each such edge is uniquely and unambiguously labeled. There are no other edges in the graph. The name of a node is the label of the edge leading from the directory node to this node. This is what is usually meant by "choosing names from a flat name-space." Social Security numbers constitute an obvious example of names using absolute naming conventions.

2.3 Relative Naming

Under the relative naming convention, the graph has unlabeled nodes but labeled edges. There is either zero or one uniquely labeled edge from any node to any other. If there is an edge labeled i from u to v, then the *distinguished name* of v

relative to u is *i*. Here, *u* is the client and *v* the named object. Names are ambiguous—a relative name is unambiguous only if qualified by some *source* node, the client node. Without additional disambiguating information, people's names are relative. One person's use of the name "John Smith" may well differ from another's.

2.4 Comparison of the Absolute and Relative Naming Conventions

2.4.1 *Locating Named Objects.* One important role of the clearinghouse is to maintain the mapping *LookUp* from names to objects. If *i* is the name of an object, then *LookUp*(i) is that object. Under the relative naming convention, *LookUp* is relative to each client node. That is, if the name of an object *v relative to u* is *i*, then $LookUp_u(i)$ is *v*. Under the absolute naming convention, *LookUp* is relative to the whole graph. That is, if the name of an object *v* is *i*, then *LookUp*(i) is *v*; we do not have to qualify *LookUp* with the source node. Thus, the database required by the absolute convention may be smaller than under the relative convention (where the number of names is on the order of the square of the number of nodes). However, since the relative convention does not require that every node directly name every other node, the domain of each *LookUp* under the relative convention will typically be much smaller than the domain for *LookUp* under the absolute convention.

The relative convention encourages decentralization, since the mapping from names to objects is relative to each node. The absolute convention encourages centralization, since there is only one mapping for the whole system. Thus the relative convention allows more efficient implementation of the *LookUp* function. Of course, one can use efficient methods such as binary search or hashing with either convention, but these make use only of *syntactic* information in names, not *semantic* information.

2.4.2 *Changing Locations or Names.* The main considerations here are the size and degree of centralization of the databases. Consider, for instance, the allocation of names. The absolute naming convention requires a centralized naming authority, allocating names for the whole graph. The relative naming convention permits decentralized naming authorities, one for each node. The local database handled by the naming authority under the relative convention will typically be much smaller than the global database handled by the naming authority under the absolute convention.

2.4.3 *Passing Names and Locations.* A major advantage of the absolute naming convention is that there is a common way for clients to refer to named objects. It is possible for any client to hand any other client the name of any object in the environment and be guaranteed that the name will mean the same thing to the second client (that is, refer to the same object). This is not the case with the relative addressing convention; if *u* and *v* are nodes, $u[i]$ need not equal $v[i]$. Under the relative naming convention, the first client must give the second client the name of the object *relative to the second client.* In practice, this means that the first client has to understand how the second client names objects. This suggests excessive decentralization; it requires too much coordination when objects are to be shared or passed.

2.5 Hierarchical Naming

Neither the absolute nor the relative naming convention is obviously superior to the other. We can do better by adding another layer of structure to the basic naming model.

We partition the graph into subgraphs, consisting of subsets of the set of nodes. We assume that each node is in exactly one subgraph. The *distinguished name* of a node is *nodename* : *subgraphname* where *subgraphname* is the name of its containing subgraph and *nodename* is the name of the node in that subgraph. This definition is well defined only if names are unambiguous within a subgraph; the absolute naming convention must be used within a subgraph. That is, within any subgraph, no two nodes can have the same name. Two different nodes may have names $A : B$ and $A : C$, however; names need be unambiguous only *within a subgraph.*

The name of a node consists of both its name within a subgraph and the name of the subgraph. As mentioned, the absolute naming convention must be used for naming nodes within any subgraph. Subgraphs are named using either the relative or the absolute naming convention.

If the absolute naming convention is used, each distinct subgraph has an unambiguous name. Since the absolute naming convention is also used for naming nodes within each subgraph, it follows that nodes have unambiguous distinguished names. Telephone numbers such as 494-4763 fit into this two-level absolute naming hierarchy. The local exchange is uniquely and unambiguously determined (within each area code) by the exchange number 494; within exchange 494, exactly one telephone has number 4763.

If the relative naming convention is used, each distinct subgraph has an unambiguous distinguished name relative to each other subgraph. And, since we are using the absolute naming convention within subgraphs, it follows that each node has an unambiguous distinguished name relative to each source. An example of this is the interface between the Xerox Grapevine mail transport mechanism [2] and the ARPANET mail transport system. A name may be *Jones.PA* within Xerox but *Jones@MAXC* outside—the subgraph name has changed.

In either case, the advantage of using a hierarchy is clear: absolute naming can be used without barring decentralization. A partitioned name suggests the search path to the object being named and encourages a decentralized naming authority.

One can imagine a hierarchy of graphs with corresponding names of the form $i_1 : i_2 : \cdots : i_k$. Examples include telephone numbers fully expanded to include country and area codes (a four-level hierarchy), or network addresses (a three-level hierarchy of network number, host number, socket number), or book-naming conventions such as the Dewey Decimal System.

For the reasons cited above, the usual distinction made between "flat" and "hierarchical" is somewhat misleading. The distinctions should be "flat" versus "relative" and "hierarchical" versus "nonhierarchical."

2.6 Abbreviations

The notion of *abbreviation* arises naturally with hierarchical naming. Within subgraph B, the name $A : B$ can be abbreviated to A without ambiguity, given the convention that abbreviations are expanded to include the name of the graph in which the client node exists. Abbreviation is a relative notion. (E.g., see [6] for another approach to abbreviations.)

2.7 Combining Networks

One major advantage of the hierarchical superstructure that we have not considered before, and that is independent of the absolute-versus-relative naming question, concerns combining networks. One feature that any clearinghouse should be able to handle gracefully is joining its database with the database of another clearinghouse, an event that happens when their respective networks are joined. For instance, consider the telephone model. When the various local telephone systems in North America combined, they did so by adding a superstructure above their existing numbering system, consisting of area codes. Area codes are the names of graphs encompassing various collections of local exchanges.

Adding new layers to names is one obvious way to combine networks. The major advantage is that, if a name is unambiguous within one network, then it is still unambiguous with its network name prefixed, even if the name also appears in some other network (because the latter name is prefixed by the name of *that* network). The major disadvantages are that the software or hardware has to be modified to admit the new level of naming and that a centralized naming authority is needed to choose names for the new layer.

The alternative to adding a new layer is expanding the existing topmost layer. For instance, the North American area code numbering system is sufficiently flexible that another area code can be added if necessary. The advantage of this is that less change is required to existing software and hardware. The disadvantage is that the interaction with the centralized naming authority, to ensure that the new area code is unambiguous, is more intimate than in the case of adding a new layer.

2.8 Levels of Hierarchy

If one chooses to use a hierarchical naming convention, an obvious question is the following: Should we agree on a constant number of levels (such as two levels in the ARPANET mailing system or four in the telephone system) or an arbitrary number of levels? If a name is a sequence of the form $i_1 : i_2 : \cdots : i_k$, should k be constant or arbitrary? There are pros and cons to either scheme. The advantage of the *arbitrary* scheme is that the naming system may evolve (acquire new levels as a result of combining networks) very easily. That is, if we now have a network with names of the form $A{:}B$ and combine this network (let us call it network C) with another network, then we can just change all our names to names of the form $A{:}B{:}C$ without changing any of the algorithms manipulating names. Allowing arbitrary numbers of levels clearly has an advantage. It also has several nontrivial disadvantages. First, all software must be able to handle an arbitrary number of levels, so software manipulating names will tend to be more complicated than in the *constant*-level scheme. Second, abbreviations become very difficult: does $A{:}B$ mean exactly that (an object with a two-level name) or is it an abbreviation for some name $A{:}B{:}C$? The disadvantage of the *constant* scheme is that one has to choose a number, and if we later add new levels, we have to do considerably more work.

2.9 Aliases

Our basic model allows each node to have exactly one name under the absolute naming convention, and exactly one name relative to any other node under the relative naming convention. An obvious extension to this model is to allow *aliases,* or alternative names, for nodes. To do this, we define an equivalence relation on names; if two names are in the same equivalence class, they are names of the same node. Under the relative naming convention, there is one equivalence relation defined on names for each client node in the graph. Under the absolute naming convention, there is only one equivalence relation for the whole graph. Each equivalence class has a distinguished member, and this we designate the *distinguished name* of the node.

The notion of aliasing is easily confused with the notion of relative naming, since each introduces multiple names for objects. The difference lies in the distinction between ambiguity and nonuniqueness. Under the relative naming convention, a name can be *ambiguous* in that it can be the name of more than one node (relative to different source nodes). Under the absolute naming convention, names are unambiguous. In either case, without aliasing, names are *unique*: if a node knows another node by name, it knows that node by exactly one name. With aliasing, names are nonunique; one node may know another by several names. Another way of expressing the difference is to consider the mapping from names to nodes. Without aliasing, the mapping is either one-to-one (under the absolute naming convention: each object has exactly one name, and no two objects have the same name) or one-to-many (under the relative naming convention: each object has exactly one name relative to any other, but many nodes may have the same name). With aliasing, the mappings become many-to-one or many-to-many.

3. CLEARINGHOUSE NAMING CONVENTION

We now describe the naming system supported by our clearinghouse. Recall first that we have a very general notion of the objects being named: an object is anything that has a name known to the clearinghouse and the intuitive property of "network visibility." We give some concrete examples in the following sections.

Objects are named in a uniform fashion. We use the same naming convention for every object, regardless of whether it is a user, a workstation, a server, a distribution list, or whatever.

A *name* is a nonnull character string of the form $\langle substring_1 \rangle : \langle substring_2 \rangle : \langle substring_3 \rangle$, where $substring_1$ denotes the *localname*, $substring_2$ the *domain*, and $substring_3$ the *organization.* Thus names are of the form $L{:}D{:}O$ where L is the localname, D the domain, and O the organization. None of the substrings may contain occurrences of ":" or "*" (the reason for the latter exclusion is given later).

Each object has a *distinguished name.* Distinguished names are absolute; no two objects may have the same distinguished name. In addition to its distinguished name, an object may have one or more aliases. Aliases are also absolute; no two objects may have the same alias. A name is either a distinguished name or an alias, but not both.

We have thus divided the world of objects into organizations and subdivided organizations into domains: a three-level hierarchy. An object is *in organization O* if it has a name of the form ⟨*anything*⟩:⟨*anything*⟩:*O*. An object is *in domain D in organization O* or *in D*:*O* if it has a name of the form ⟨*anything*⟩:*D*:*O*.

This division into organizations and, within them, domains, is a logical rather than a physical division. An organization will typically be a corporate entity such as Xerox Corporation. The names of all objects within Xerox will be of the form ⟨*anything*⟩:⟨*anything*⟩:*Xerox*. Xerox will choose domain names to reflect administrative, geographical, functional, or other divisions. Very large corporations may choose to use several organization names if their name space is very large. In any case, the fact that two addressable objects have names in the same domain or organization does not necessarily imply in any way that they are physically close.

3.1 Rationale

We use a uniform naming convention for all objects, regardless of their type.

Objects known to the clearinghouse have absolute distinguished names and aliases. Thus we favor an absolute naming convention over a relative naming convention. Most systems (including most mail transport systems) have opted for a relative naming convention. However, the advantages of an absolute convention are so clear that we are willing to put up with the burden of some centralization. By choosing the naming convention carefully, we can reduce the pain of this centralization to an acceptable level.

Names are hierarchical. We rejected a nonhierarchical system because, among their other advantages, hierarchical names can be used to help suggest the search path to the mapping.

We have chosen a three-level naming hierarchy, consisting of *organizations*, within them *domains*, and within them *localnames*. We did not choose the arbitrary-level scheme because of the greater complexity of the software required to handle names, because we do not think that networks will be combined very often, and because (as with area codes) we make the name space for organizations large enough so that combinations can generally be made within the three-level hierarchy by merging two sets of existing ones. We choose three levels, rather than, say, two or four, for pragmatic reasons. A mail system such as Grapevine [2] works well with only a two-level hierarchy, combining networks across the company's divisional boundaries. We add the third level primarily to facilitate combining networks across company lines. However, the clearinghouse does not give any particular meaning to the partitions; this is why we chose the relatively innocuous names "organization" and "domain."

4. USER NAMES

One important class of "objects" known to the clearinghouse is the set of users. For instance, the clearinghouse may be used to map a user's name into the network address of the mail server where his mailbox resides. To deliver a piece of mail to a user, an electronic mail system first asks the clearinghouse where the mailbox for that user is and then routes the piece of mail to that server.

A major design decision is how the localnames of users are chosen. We describe our approach to naming users, as this will provide further motivation for our naming convention. The following is not part of the design of our clearinghouse but illustrates one of its important uses.

A *User Name* is a string of the form

$\langle firstname\rangle\langle blanks\rangle\langle middlename\rangle\langle blanks\rangle\langle lastname\rangle:\langle domain\rangle:\langle organization\rangle.$

Here, ⟨*firstname*⟩, ⟨*middlename*⟩, and ⟨*lastname*⟩ are strings separated by blanks (they may themselves contain blanks, as in the last name *de Gaulle*). ⟨*firstname*⟩, ⟨*middlename*⟩, and ⟨*lastname*⟩ are the first name, middle name, and last name of the user being named. The following are examples of user names:

David Stephen Jones : *SDD* : *Xerox*
John D. Smith : *SDD* : *Xerox*

The basic scheme, therefore, is that a name consists of the user's three-part localname, domain, and organization. The localname is the person's legal name. The reason for making the user name the complete three-part name (rather than just the last name) is to discourage clashes of names and encourage unambiguity. The chance of there being two people with the name *David Stephen Jones* in domain *SDD* in organization *Xerox* is, one hopes, rather remote, and certainly more remote than there being two people with last name *Jones*.

Our convention for naming users differs from those used in most computer environments in requiring that names be absolute and in using full names to reduce the chance of ambiguity. We have discussed the issue of absolute-versus-relative naming conventions already, but the second topic deserves attention because it shows the advantages of having a consistent approach to aliases.

The most common way of choosing unambiguous user names in computer environments is to use last names prefixed with however many letters are needed to exclude ambiguity. Thus, if there are two *Jones*'s, one might be *DJones* and the other *HJones*. This scheme we find unsatisfactory. It is difficult for users (who have to remember to map their name for the person into the system's name for the person) and difficult for system administrators (who have to manage this rather artificial scheme). Further, it requires users to change their system names occasionally: if a system name is presently *DJones* and other *D. Jones* becomes a user, the system name must be changed to avoid ambiguity.

Our convention is not cumbersome to the user, since we use the same firstname-middlename-lastname convention people are used to already. However, since users would find it very cumbersome to type in full names, various aliases for user names are stored in the clearinghouse. For instance, associated with the user name *David Stephen Jones* might be the aliases *David Jones*, *D. Jones*, and *Jones*. Since our naming convention requires that aliases be absolute, it follows that no two users can have the same alias.

4.1 Birthmarks

Even with our convention of using a user's full name, there is a possibility that there will be two users with exactly the same name in a domain. Our approach is to disallow this and let the two users (or a system administrator) choose unambiguous names for each. Another approach is to add as a suffix to each full name a "birthmark." A ⟨*birthmark*⟩ is any string that, together with the user name, the domain name, and the organization name, unambiguously identifies the user. The birthmark may be a universal identifier (perhaps the concatenation of the processor number of the workstation on which the name is being added together with the time of day). It might be the Social Security number of the individual (perhaps not a good idea on privacy grounds). It might be just a positive integer; the naming authority for each domain is responsible for handing out integers. In any case, the combination of the full name and the birthmark must be unambiguous so that no two users can have the same legal name. In the case that a birthmark is not meaningful to humans, ambiguities must be resolved by providing users of such names with additional information such as a "title." The mappings described next provide a mechanism for supplying such disambiguating comments.

5. MAPPINGS

Now that we know how to name the objects known to the clearinghouse, we treat the question of what names are mapped into.

The clearinghouse maps each name into a set of *properties* to be associated with that name. A *property* is an ordered tuple consisting of a *PropertyName*, a *PropertyType*, and a *PropertyValue*. The clearinghouse maintains mappings of the form:

$$\mathit{name} \rightarrow \{\langle \mathit{PropertyName}_1, \mathit{PropertyType}_1, \mathit{PropertyValue}_1\rangle, \ldots, \langle \mathit{PropertyName}_k, \mathit{PropertyType}_k, \mathit{PropertyValue}_k\rangle\}.$$

More precisely, to admit aliasing, the clearinghouse maps equivalence classes, rather than names, into sets of properties. Each equivalence class consists of a distinguished name and its aliases. The value of k is not fixed for any given name. A name may have associated with it any number of properties.

A *PropertyValue* is a datum of type *PropertyType*. There are only two types of property values. The first, of type *item*, is "uninterpreted block of data." The clearinghouse attaches no meaning to the contents of this datum but treats it as just a sequence of bits. The second, of type *group*, is "set of names." A name may appear only once in the set, but the set may contain any number of different names (including aliases and names of other groups). The names "item" and "group" reflect the semantics attached by the clearinghouse, whether the property is an individual datum or a group of data.

Mapping a name into a network address is an example of a type *item* mapping, as in

$$\mathit{Daisy{:}SDD{:}Xerox} \rightarrow \{\langle \mathit{Printer, item, network\ address\ of\ the\ printer\ named\ Daisy}\rangle\}.$$

A distribution list in electronic mail is an example of a mapping of type *group*, as in

$$\mathit{CHAuthors{:}SDD{:}Xerox} \rightarrow \{\langle \mathit{Distribution\ List, group}, \{\text{“}\mathit{Dalal{:}SDD{:}Xerox}\text{”}, \text{“}\mathit{Oppen{:}SDD{:}Xerox}\text{”}\}\rangle\}.$$

Many properties may be associated with a name, as in

> *John D. Smith*:*SDD*:*Xerox*
> {⟨*User, item, descriptive comment such as V.P. Marketing*⟩,
> ⟨*Password, item, password to be used for user authentication*⟩,
> ⟨*FileServerName, item, name of file server containing user's files*⟩,
> ⟨*Mailbox, item, name of mail server where user's mail is stored*⟩,
> ⟨*PrinterNames, group, set of names of local printers any of which may be used*⟩}.

In this example, the clearinghouse is used to store the user's "profile." Note that we choose to map the user's name into the *name* of his local file server (and mailbox and printer) rather than directly into its network address. The reason for this extra level of indirection is that the name of the file server will perhaps never change, but its location certainly will occasionally change, and we do not want a change in a server's location to require a major update of the clearinghouse's database.

5.1 Rationale

Objects tend to fall into two broad categories: objects such as workstations, servers, or people whose names are mapped into descriptions, and objects such as distribution lists whose names are mapped into sets of names.

We differentiate between properties of type item and properties of type group but allow many properties of differing types to be associated with each name. The above example giving the mapping for a user name shows why. Unlike the simpler telephone model, where a single mapping from a user name into a telephone number suffices, we want to map a user's name into a richer collection of information. This applies even to nonuser individuals. We may want to associate with a printer's name not only its location (so that files to be printed can be sent to it), but also information describing what fonts the printer supports, whether it prints in color, and so on.

The main reason for having "set of names" as a distinct data type is to allow different clients to update the same set simultaneously. For instance, if the set represents an electronic mail distribution list, we allow two users to add themselves to this list asynchronously.

5.2 Generic Names

The set of property names known to the clearinghouse defines a set of generic names by which the clearinghouse provides a Yellow Pages-like facility. Such a capability can be used as follows.

Client software can request a service in a standardized fashion and need not remember what named resources are available. For instance, each user workstation generally has a piece of software that replies to the user command "Help!" This software accesses some server to obtain the information needed to help the user. Suppose the generic name "Help Service" is agreed upon as the standard property name for such a service. To find the addresses of the servers providing help to users in *SDD*:*Xerox*, the workstation software asks to list all objects of name "*:*SDD*:*Xerox*" with propertyname *Help Service*. The "wild card" character "*" matches zero or more characters. This piece of code can be used by any workstation, regardless of its location.

The "wild card" feature allows clients to find valid names where they have only partial information or can only guess the name. It is particularly useful in electronic mail and in other uses of user names.

If looking up "*Smith*" with propertyname *Mailbox* fails, because "*Smith*" is ambiguous, the electronic mail system may choose to list all names of the form "*Smith*:*SDD*:*Xerox*" with propertyname *User* to find the set of user names matching this name. It presents this set to the sender of the mail and allows him to choose which unambiguous name is appropriate. A simple algorithm to use in general might be to take any string provided by the user, surround the string with *'s, delete any periods, and replace any occurrence of ⟨blank⟩ by *⟨blank⟩. Thus *David S. Jones* becomes **David* S* Jones**, which matches *David Stephen Jones*, as desired.

6. THE CLIENT'S PERSPECTIVE

Recall first that the clients of the clearinghouse are pieces of software and hardware making use of the clearinghouse/client interface. The fact that people are not clients of the clearinghouse (except indirectly by means of a software interface) immediately introduces an important difference between our clearinghouse and the telephone system's. The telephone system relies on human judgment and human interaction. The clients of our clearinghouse are machines, not people, and so all aspects of client-clearinghouse interaction, including fault tolerance, must be fully automated.

The clearinghouse (and its associated database) is decentralized and replicated. That is, instead of one *global* clearinghouse, there are many *clearinghouse servers* scattered throughout the internetwork (perhaps, but not necessarily, one per local network), each storing a copy of a portion of the global database. Decentralization and replication increase efficiency (it is faster to access a clearinghouse server physically nearby), security (each organization can control access to its own clearinghouse servers), and reliability (if one clearinghouse server is down, perhaps another can respond to a request). However, we do assume that there is one *global* database (i.e., conceptually; physically, the database is decentralized). Each clearinghouse server contains a portion of this database. We make no assumptions about how much of the database any particular clearinghouse server stores. The union of all the local databases stored by the clearinghouse servers is the global database.

A client of the clearinghouse may refer by name to, and query the clearinghouse about, any named object in the distributed environment (subject to access control) regardless of the location of the object, the location of the client, or the present distributed configuration of the clearinghouse. We make no assumptions about the physical proximity of clients of the clearinghouse to the objects whose names they present to the clearinghouse. A request to the clearinghouse to bind a name to its properties may originate anywhere in the internetwork. This makes the internal structure of our clearinghouse considerably more intricate than that of the telephone clearinghouse (where clients have to know which local telephone directory to access) but makes it much easier to use.

In order to provide a uniform way for clients to access the clearinghouse, we assume that all clients contain a (generally very small) clearinghouse component, which we call a *clearinghouse stub*. Clearinghouse stubs contain pointers to clearinghouse servers, and they provide a uniform way for clients to access the clearinghouse.

A client requests a binding from its stub clearinghouse. The stub communicates with clearinghouse servers to get the information. A client of the clearinghouse stub need not concern itself with the question of which clearinghouse server actually contains the binding—the client's stub in conjunction with the clearinghouse servers automatically finds the mapping if it exists.

Updates to the various copies of a mapping may occur asynchronously and be interleaved with requests for bindings of names to properties. Therefore, clearinghouse server databases may occasionally have incorrect information or be mutually inconsistent. (In this respect, we follow the telephone system's model and not the various models for distributed databases in which there is a notion of "indivisible transaction." We find the latter too complicated for our needs.) Therefore, as in the telephone system, bindings given by clearinghouse servers should be considered by clients to be *hints*. If a client requests the address of a printer, it may wish to check with the server at that address to make sure it is in fact a printer. If not, it must be prepared to find the printer by other means (perhaps the printer will respond to a local broadcast of its name), wait for the clearinghouse to receive the update, or reject the printing request. If the information given out by the clearinghouse is incorrect, it cannot, of course, guarantee that the error in its database will be corrected. It can only hope that whoever has invalidated the information will send (or preferably already has sent) the appropriate update. However, the clearinghouse does guarantee that any inconsistencies between copies of the same portion of the database will be resolved and that any such inconsistency is transient. This guarantee holds even in the case of conflicting updates to the same piece of information; the clearinghouse arbitrates between conflicting updates in a uniform fashion.

Assuming this model of goodwill on the part of its clients—that they will quickly update any clearinghouse entry they have caused to become invalid—and assuming an automatic arbitration mechanism for quickly resolving in a predictable fashion any transient inconsistencies between clearinghouse servers, clients can assume that any information stored by the clearinghouse either is correct or, if not, will soon be corrected. Clients, therefore, may assume that the clearinghouse either contains the *truth* about any entry or soon will contain it. It is very important that clients can trust the clearinghouse in this way, because the clearinghouse is often the only source of information available to the client on the locations of servers, on user profiles, and so on.

The fact that the information returned by the clearinghouse is treated by the clients as both the truth (the information is available only from the clearinghouse and so had better be correct) and a hint (the information may be temporarily incorrect) is not self-contradictory. It merely reflects the difference between the long-term and short-term properties of clearinghouse information.

6.1 Binding Strategies

An important consideration to be taken by the client is that of *when* to ask the clearinghouse for a binding. The binding technique used greatly influences the ability of the system to react to changes in the environment.

There are three possibilities: *static* binding, in which names are bound at the time of system generation; *early* binding, in which names are bound, say, at the time the system is initialized; and *late* binding, in which names are bound at the time their bindings are to be used. (The boundaries between the three possibilities are somewhat ill-defined; there is a continuum of choices.) The main trade-off to be taken into consideration in choosing a binding strategy is *performance* versus *flexibility*.

The later a system binds names, the more gracefully it can react to changes in the environment. If client software binds names statically, the software must be updated whenever the environment changes. For instance, if software supporting printing directly stores the addresses of the print servers (that is, uses a static binding strategy), it must be updated whenever new print servers are added or existing servers are moved or removed. If the software uses a late binding strategy, it will automatically obtain the most up-to-date bindings known to the clearinghouse.

On the other hand, binding requires the resolution of one or more indirect references, and this takes time. Static or early binding increases run-time efficiency since, with either, names are already bound at run-time. Further, late binding requires interaction with the clearinghouse at run-time. Although we have designed the clearinghouse to be very reliable, the possibility exists that a client may occasionally be unable to find any clearinghouse server up and able to resolve a reference.

There are, therefore, advantages and disadvantages to any binding strategy. A useful compromise combines early and late binding, giving the performance and reliability of the former and the flexibility of the latter. The client uses early binding wherever possible and late binding only if any of these (early) bindings becomes invalid. Thus, software supporting printing stores the addresses of print servers at initialization, and updates these addresses only if they become invalid. Of course, the client must be able to recognize if a stored address is invalid (just as it must accept the possibility that the information received from the clearinghouse is temporarily invalid). We discuss hint validation further in the appendix.

7. CLIENT INTERFACE

The clearinghouse provides a basic set of operations, some of which are exported operations, which may be called by clients of the clearinghouse by means of the *clearinghouse stub* resident in the client, and some of which are internal operations used by clearinghouse components to communicate with each other. Strictly speaking, the clearinghouse requires only a very few commands, for reading, adding, and deleting entries. We provide many different operations, however, as described in detail in [10].

We give different commands for different types (for instance, different commands to add an item and to add a group) to provide a primitive type-checking facility.

We give different operations for different levels of granularity (e.g., different commands for adding groups and adding elements to a group) for three reasons. First, this minimizes the data that must be transmitted by the clearinghouse or the client when reading or updating an entry. Second, different clients are permitted to change different parts of the same entry at the same time. For instance, two clients may add different elements to the same group simultaneously; if each were required to update the whole entry, their two updates would conflict. Third, we make use of the different operations for different levels of granularity in our access control facility.

Finally, we provide separate operations for changing a propertyvalue, although these operations are functionally equivalent to deleting the original and adding the new one. However, changing an entry constitutes one indivisible transaction; deleting and adding an entry constitute two transactions separated by a time period during which another client may try to read the incorrectly empty entry.

8. CLEARINGHOUSE STRUCTURE

We now describe how the clearinghouse is structured internally.

8.1 Clearinghouse Servers

The database of mappings is decentralized. Copies of portions of the database are contained in *clearinghouse servers*, which are servers spread throughout the internetwork. We refer to the union of all these clearinghouse servers as "the clearinghouse." Each clearinghouse server is a named object in the internetwork and so has a distinguished name and possibly aliases as well.

Every client of the clearinghouse contains a clearinghouse component, called a *clearinghouse stub*. Clearinghouse stubs provide a uniform way for clients to access the clearinghouse. Stub clearinghouses do not have names (although they will typically be on machines containing named objects). Stubs are required to find at least one clearinghouse server (e.g., by local or directed broadcast [3]).

8.2 Domain and Organization Clearinghouses

Corresponding to each domain D in each organization O are one or more clearinghouse servers each containing a copy of all mappings for every name of the form $\langle anything \rangle{:}D{:}O$. Each such clearinghouse server is called a *domain clearinghouse for* $D{:}O$. (Each clearinghouse server that is a domain clearinghouse for $D{:}O$ may contain other portions of the database other than just the database for this domain, and each of the domain clearinghouses for $D{:}O$ may differ on what other portions of the global database, if any, they contain.) There is at least one domain clearinghouse for each domain in the distributed environment. Domain clearinghouses are addressable objects in the internetwork and hence have names. Each domain clearinghouse for each domain in organization O has a name of the form $\langle anything \rangle{:}O{:}CHServers$, which maps into the network address of the server under property name *CH Location*. (*CHServers* is a reserved organization name.) Thus, if $L{:}O{:}CHServers$ is the name of a domain clearinghouse for $D{:}O$, then there is a mapping of the form

$$L: O: CHServers \rightarrow \{\ldots, \langle CH\ Location,\ item,\ network\ address \rangle, \ldots\}.$$

For each domain $D:O$, we require that $D:O:\mathit{CHServers}$ map into the set of names of domain clearinghouses for $D:O$ under property name *Distribution List.* For example, if the domain clearinghouses for domain $D:O$ have names $L_1:O:\mathit{CHServers}, \ldots, L_k:O:\mathit{CHServers}$, then there is a mapping of the form

$$D:O:\mathit{CHServers} \rightarrow \{\ldots, \langle \mathit{Distribution\ List, group}, \{L_1:O:\mathit{CHServers}, \ldots, L_k:O:\mathit{CHServers}\}\rangle, \ldots\}.$$

For each i and j, $L_i:O:\mathit{CHServers}$ is a *sibling* of $L_j:O:\mathit{CHServers}$ *for domain* $D:O$. Thus, we have given a name to the set of sibling domain clearinghouses for each domain in organization O.

The names of all domain clearinghouses and all sets of sibling domain clearinghouses for domains in organization O are themselves names in the reserved domain $O:\mathit{CHServers}$. We call each domain clearinghouse for this reserved domain an *organization clearinghouse for O* since it contains the name and address of every domain clearinghouse in the organization. In particular, if $L_1:O:\mathit{CHServers}, \ldots, L_k:O:\mathit{CHServers}$ are the domain clearinghouses for any domain $D:O$, then each organization clearinghouse for O contains the mappings

$$\begin{aligned}
&D:O:\mathit{CHServers} \rightarrow \{\ldots, \langle \mathit{Distribution\ List, group}, \{L_1:O:\mathit{CHServers}, \ldots, L_k:O:\mathit{CHServers}\}\rangle, \ldots\},\\
&L_1:O:\mathit{CHServers} \rightarrow \{\ldots, \langle \mathit{CH\ Location, item, network\ address}\rangle, \ldots\},\\
&\quad\vdots\\
&L_k:O:\mathit{CHServers} \rightarrow \{\ldots, \langle \mathit{CH\ Location, item, network\ address}\rangle, \ldots\}.
\end{aligned}$$

Since $O:\mathit{CHServers}$ is a domain, there is at least one domain clearinghouse for $O:\mathit{CHServers}$ and hence at least one organization clearinghouse for O. Each such clearinghouse has a name of the form $\langle \mathit{anything}\rangle:\mathit{CHServers}:\mathit{CHServers}$, which maps into the network address of the server under property name *CH Location.* Thus, if $L:\mathit{CHServers}:\mathit{CHServers}$ is the name of a domain clearinghouse for $O:\mathit{CHServers}$ (that is, an organization clearinghouse for O), then there is a mapping of the form

$$L:\mathit{CHServers}:\mathit{CHServers} \rightarrow \{\ldots, \langle \mathit{CH\ Location, item, network\ address}\rangle, \ldots\}$$

For each organization O, we require that $O:\mathit{CHServers}:\mathit{CHServers}$ map into the set of names of organization clearinghouses for O under property name *Distribution List.* For example, if the organization clearinghouses for O have names $L_1:\mathit{CHServers}:\mathit{CHServers}, \ldots, L_k:\mathit{CHServers}:\mathit{CHServers}$, then there is a mapping of the form

$$\begin{aligned}
&O:\mathit{CHServers}:\mathit{CHServers} \rightarrow\\
&\quad \{\ldots, \langle \mathit{Distribution\ List, group},\\
&\qquad \{L_1:\mathit{CHServers}:\mathit{CHServers}, \ldots, L_k:\mathit{CHServers}:\mathit{CHServers}\}\rangle, \ldots\}.
\end{aligned}$$

Each $L_i:\mathit{CHServers}:\mathit{CHServers}$ is called a *sibling* of $L_j:\mathit{CHServers}:\mathit{CHServers}$ *for organization O.* Thus, we have given names to the set of sibling organization clearinghouses for each organization O.

Note that each organization clearinghouse for *O* points directly to every domain clearinghouse for any domain in *O*, and hence indirectly to every object with a name in *O*.

Note the distinction between *clearinghouse servers* on the one hand and *domain* and *organization* clearinghouses on the other. The former are physical entities that run code, contain databases, and are physically resident on network servers. The latter are logical entities and are a convenience for referring to the clearinghouse servers that contain specific portions of the global database. A particular clearinghouse server may be a domain clearinghouse for zero or more domains and an organization clearinghouse for one or more organizations.

8.3 Interconnections between Clearinghouse Components

Organization clearinghouses point "downward" to domain clearinghouses, which point "downward" to objects. Further interconnection structure is required so that stub clearinghouses can access clearinghouse servers and clearinghouse servers can access each other.

Each clearinghouse server is required to be an organization clearinghouse for the reserved organization *CHServers*, and hence each clearinghouse server points "upward" to *every* organization clearinghouse. In this way, each clearinghouse server knows the name and address of every organization clearinghouse.

We do not require a clearinghouse server to keep track of which clearinghouse stubs point to it, so these stubs will not be told if the server changes location and must rely instead on other facilities, such as local or directed broadcast, to find a clearinghouse server if the stored address becomes invalid.

8.4 Summary

Each clearinghouse server contains mappings for a subset of the set of names. If it is a domain clearinghouse for domain *D* in organization *O*, it contains mappings for all names of the form ⟨*anything*⟩ : *D* : *O*. If it is an organization clearinghouse for organization *O*, it contains mappings for all names of the form ⟨*anything*⟩ : *O*:*CHServers* (names associated with domains in *O*). Each clearinghouse server contains the mappings for all names of the form ⟨*anything*⟩ : *CHServers* : *CHServers* (names associated with organizations); that is, the database associated with the reserved domain *CHServers* : *CHServers* is replicated in every clearinghouse server. Stubs point to any clearinghouse server.

For every domain *D* in an organization *O*, *D*:*O*:*CHServers* names the set of names of sibling domain clearinghouse servers for *D*:*O*. For every organization *O*, *O*:*CHServers*:*CHServers* names the set of names of sibling organization clearinghouse servers for *O*. The name *CHServers*:*CHServers*:*CHServers* contains the set of names of *all* clearinghouse servers. (We do not make use of this name in this paper.)

This clearinghouse structure allows a relatively simple algorithm for managing the decentralized clearinghouse. However, it does require that copies of the mappings for all names of the form ⟨*anything*⟩:*CHServers*:*CHServers* be stored in *all* clearinghouse servers.

9. DISTRIBUTED LOOKUP ALGORITHM

Suppose that a stub clearinghouse receives the query to look up an item. The stub clearinghouse follows the general protocol:

The stub clearinghouse contacts any clearinghouse server and passes it the query. If the clearinghouse server that receives the stub's query is a domain clearinghouse for $B : C$, it can immediately return the answer to the stub, which in turn returns it to the client. Otherwise, the clearinghouse server returns the names and addresses of the organization clearinghouses for C, which it is guaranteed to have. The stub contacts any of these clearinghouse servers. If this clearinghouse server happens also to be a domain clearinghouse for $B : C$, it can immediately return the answer to the stub, which in turn returns it to the client. Otherwise, the clearinghouse server returns the names and addresses of the domain clearinghouses for $B : C$, which it is guaranteed to have. The stub contacts any of these, since any of them is guaranteed to have the answer. An algorithmic description of this search process can be found in [10].

The domain clearinghouse for $B : C$ that returns the answer to the query does so after authenticating the requestor and ascertaining that the requestor has appropriate access rights.

In the worst case, a query conceptually moves "upward" to a domain clearinghouse, to an organization clearinghouse, and then "downward" to one of that organization's domain clearinghouses. The number of clearinghouse servers that a stub has to contact will never exceed three: the clearinghouse server whose address it knows, an organization clearinghouse for the organization containing the name in the query, and a domain clearinghouse in that organization.

However, before sending the query "upward," each clearinghouse component *optionally* first sees if it can shortcut the process by sending the query "sideways," cutting out a level of the hierarchy. (This is similar to the shortcuts used in the routing structure of the telephone system.) These "sideways" pointers are cached pointers, maintained for efficiency. For instance, consider domain clearinghouses for *PARC* and for *SDD*, two logical domains within organization *Xerox*. Depending on the traffic, it may be appropriate for the *PARC* clearinghouse to keep a direct pointer to the *SDD* clearinghouse, and vice versa. This speeds queries that would otherwise go through the *Xerox* organization clearinghouse. (Cached entries are not kept up to date by the clearinghouse. This is not a serious problem; if a cached entry is wrong, it is deleted, and the general algorithm described above is used instead.)

To increase the speed of response even further, each clearinghouse server could be a domain clearinghouse for both domains. Alternatively, if the number of domains in *Xerox* is relatively small, it may be appropriate to make each clearinghouse server in *Xerox* an organization clearinghouse for *Xerox*. In this way, each clearinghouse server in *Xerox* always points to every other clearinghouse server in *Xerox*.

Local queries (that is, queries about names "logically near" the clearinghouse stub) will typically be answered more quickly than nonlocal queries. That is appropriate. The caching mechanism (for storing of "sideways" pointers) can be used to fine-tune clearinghouse servers to respond faster to nonlocal but frequent queries.

10. DISTRIBUTED UPDATE ALGORITHM

The distributed update algorithm we use to alter the clearinghouse database is closely related to the distributed update algorithm used by Grapevine's registration service [2].

The basic model is quite simple. Assume that a client wishes to update the clearinghouse database. The request is submitted via the stub resident in the client. The stub contacts any domain clearinghouse containing the mapping to be updated. The domain clearinghouse updates its own database and acknowledges that it has done so. The interaction with the client is now complete. The domain clearinghouse then propagates the update to its siblings if the database for this domain is replicated in more than one server.

The propagation of updates is not treated as an indivisible transaction. Therefore, sibling clearinghouse servers may have databases that are temporarily inconsistent; one server may have updated its database before another has received or acted on an update request. This requires mechanisms for choosing among conflicting updates and dealing with out-of-order requests. The manner in which the clearinghouse deals with these issues in the context of the services it provides can be found in [10].

Since distributed office information systems usually have an electronic mail delivery facility that allows "messages" to be sent to recipients that are not ready to receive them [2], we make use of this facility to propagate updates. Clearinghouse servers send their siblings timestamped update messages (using the appropriate distribution lists reserved for clearinghouse servers) telling them of changes to the databases. There is a possible time lag between the sending and the receipt of a message. Since our clearinghouse design does not require that updating be an indivisible process, this is not a problem.

11. SECURITY

We restrict ourselves to a brief discussion of two issues. The first concerns protecting the clearinghouse from unauthorized access or modification and involves *authentication* (checking that you are who you say you are); the second concerns *access control* (checking that you have the right to do what you want to do). We do not discuss how the network ensures (or does not ensure) secure transmission of data, or how two mutually suspicious organizations can allow their respective clearinghouse servers to interact and still keep them at arm's length.

11.1 Authentication

When a request is sent by a client to a clearinghouse server to read from or write onto a portion of the clearinghouse database, the request is accompanied by the credentials of the client. If the request is an internal one, from one clearinghouse server to another, the requestor is the name of the originating clearinghouse server and carries the credentials of the originating clearinghouse server. The clearinghouse thus makes sure that internally generated updates come only from "trusted" clearinghouse servers. The clearinghouse uses a standard authentication service to handle authentication, and the authentication service uses the clearinghouse to store its authentication information [9].

11.2 Access Control

Once a client has been authenticated, it is granted certain privileges. Access control is provided at the domain level and at the property level, and not at the mapping (set of properties) level nor at the level of element of a group. Associated with each domain and each property is an *access control list*, which is a set of the form

$$\{\langle \textit{set of names}_1, \textit{set of operations}_1\rangle, \ldots, \langle \textit{set of names}_k, \textit{set of operations}_k\rangle\}.$$

Each tuple consists of a set of names and the set of operations each client in the set may call. Access control is described in [10].

Certain operations that modify the clearinghouse database are protected only at the domain level. These are operations that are typically executed only by *domain system administrators* and by other clearinghouse servers. Examples of such operations are adding or deleting a new name or alias and adding or deleting an item or group for a name. Other operations such as looking up an item or adding a name to a group are protected at the property level.

12. ADMINISTRATION

An internetwork configuration of several thousand users and their associated workstations, printers, file servers, mail servers, etc., requires considerable management. Administrative tasks include managing the name and property name space; bringing up new networks; deciding how to split an organization into domains (reflecting administrative, geographical, functional, or other divisional lines); deciding which objects (users, services, etc.) belong to which domains; adding, changing, and deleting services (such as mail services, file services, and even clearinghouse services); adding and deleting users; maintaining users' passwords, the addresses of their chosen local printers, mail and file servers, and so on; and maintaining access lists and other security features of the network. We have designed the clearinghouse so that system administration can be decentralized as much as possible.

An algorithmic description of the process by which a new organization or domain clearinghouse server is added to the internetwork can be found in [10].

13. CONCLUSIONS

A powerful binding mechanism that brings together the various network-visible objects of a distributed system is an essential component of any large network-based system. The clearinghouse provides such a mechanism.

Since we do not know how distributed systems will evolve, we have designed the clearinghouse to be as open-ended as possible. We did not design the clearinghouse to be a general-purpose, relational, distributed database or a distributed file system, although the functions it provides are superficially similar. It is not clear that network binding agents, relational databases, and file systems should be thought of as manifestations of the same basic object; their implementations appear to require different properties. In any case, we certainly did not try to solve the "general database problem," but instead attempted to design a system that is implementable now within the existing technology and yet can evolve as distributed systems evolve.

A phased implementation of this design is currently underway. Xerox's Network System product may choose to deviate in minor ways from this design and enforce different administrative policies than those described here. A future paper will describe the network protocols used in implementing the clearinghouse and our experiences with them.

APPENDIX. NETWORK ADDRESSES AND ADDRESS VERIFICATION

In Xerox's Network System, a *network address* is a triple consisting of a *network number*, a *host number*, and a *socket number* [5]. There is no clear correspondence between machines and the addressable objects known to the clearinghouse. One machine on an internetwork may contain many named objects; for instance, a machine may support a file service and a printer service (a *server* may contain many *services*). These different objects resident on the same machine may use the same network address even though they are separate objects logically and have different names. This introduces no problems since the clearinghouse does not check for uniqueness of addresses associated with names. Alternatively, different objects physically resident on one machine may have different network addresses, since a machine may contain many different socket numbers. To allow both possibilities, we map the names of addressable objects into network addresses without worrying about the configurations of the machines in which they are resident.

However, it may be that one machine has more than one network address, since it may be physically part of more than one network. Therefore, the name of an addressable object such as printer or file server may be mapped into a set of network addresses, rather than a single address. However, these addresses may differ only in their network numbers; objects may be physically resident in one machine only.

Since the addresses given out by the clearinghouse may be momentarily incorrect, clients need a way to check the accuracy of the network addresses given out by the clearinghouse. One way is to insist that each addressable object have a *uniqueid*, an absolute name which might, for example, consist of a unique processor number (hardwired into the processor at the factory) concatenated to the time of day. The uniqueid is used to check the accuracy of the network addresses supplied by the clearinghouse. This uniqueid is stored with the network addresses in the clearinghouse. When a client receives a set of addresses from the clearinghouse that are allegedly the addresses of some object, it checks with the object to make sure the uniqueid supplied by the clearinghouse agrees with the uniqueid stored by the object.

In summary, in the Xerox internetwork environment, the address of an object is stored as a tuple consisting of a set of network addresses and a uniqueid.

ACKNOWLEDGMENTS

Many people have examined the requirements and structure of clearinghouse-like systems; and we have profited from the earlier efforts of Steven Abraham, Doug Brotz, Will Crowther, Bob Metcalfe, Hal Murray, and John Shoch. The organization of the clearinghouse and, in particular, the lookup and updating algorithms have been very heavily influenced by the work of the Grapevine project; we thank Andrew Birrell, Roy Levin, and Mike Schroeder for many stimulating discussions. The clearinghouse is a fundamental component of Xerox's Network System product line, and we are indebted to Marney Beard, Charles Irby, and Ralph Kimball for their considerable input on what the Star workstation needed from the clearinghouse. Dave Redell, who is responsible for the Network System electronic mail system, was a constant source of inspiration and provided us with many useful ideas. Bob Lyon and John Maloney implemented the clearinghouse; we thank them for turning the clearinghouse design into reality.

REFERENCES

(Note: References, 4, 7, 8, 13, and 16 are not cited in the text.)

1. Abraham, S.M., and Dalal, Y.K. Techniques for decentralized management of distributed systems. In 20th IEEE Computer Society International Conference (Compcon) (San Francisco, Feb. 1980), pp. 430–436.
2. Birrell, A.D., Levin, R., Needham, R.M., and Schroeder, M.D. Grapevine: An exercise in distributed computing. *Commun. ACM 25*, 4 (Apr. 1982), 260–274.
3. Boggs, D.R. Internet Broadcasting. Ph.D. dissertation, Electrical Eng. Dept., Stanford Univ., Stanford, Calif., Jan. 1982.
4. Boggs, D.R., Shoch, J.F., Taft, E.A., and Metcalfe, R.M. PUP: An internetwork architecture. *IEEE Trans. Commun., COM-28,* 4 (Apr. 1980), 612–624.
5. Dalal, Y.K. Use of multiple networks in Xerox's Network System, *Computer 15,* 8 (Aug. 1982), 10–27.
6. Daley, R.C., and Neumann, P.G. A general-purpose file system for secondary storage. In *Proc. Fall Joint Computer Conf.,* Vol. 27, pt. 1. Spartan, Washington, D.C., 1965, pp. 213–228.
7. The Ethernet, a local area network: Data link layer and physical link layer specifications, Version 1.0, Xerox Office Systems Div., Palo Alto, Calif., Sept. 30, 1980.
8. Metcalfe, R.M., and Boggs, D.R. Ethernet: Distributed packet switching for local computer network. *Commun. ACM 19,* 7 (July 1976), 395–404.
9. Needham, R.M., and Schroeder, M.D. Using encryption for authentication in large networks of computers. *Commun. ACM 21,* 12 (Dec. 1978), 993–999.
10. Oppen, D.C., and Dalal, Y.K. The Clearinghouse: A decentralized agent for locating named objects in a distributed environment. Rep. OPD-T8103, Xerox Office Systems Division, Palo Alto, Calif., Oct. 1981.
11. Pickens, J.R., Feinler, E.J., and Mathis, J.E. The NIC name server—A datagram based information utility. In Proceedings 4th Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, Calif., Aug. 1979, pp. 275–283.
12. Saltzer, J.H. On the naming and binding of network destinations. In Proceedings IFIP TC 6 Symposium on Local Networks (Florence, Italy, Apr. 1982), pp. 311–317.
13. Schickler, P. Naming and addressing in a computer-based mail environment. *IEEE Trans. Commun. COM-30* 1 (Jan. 1982), xxx.
14. Shoch, J.F. Internetwork naming addressing and routing. In 17th IEEE Computer Society International Conference (Compcon) (San Francisco, Sept. 1978), pp. 72–79.
15. Solomon, M., Landweber, L.H., and Neuhengen, D. The CSNET name server. *Computer Networks 6,* 3 (July 1982), 161–172.
16. Xerox Corporation. Office systems technology. Rep. OSD-R8203, Xerox Office Systems Division, Palo Alto, Calif., Nov. 1982.

Received August 1982; revised April 1983; accepted April 1983